(19)日本国特許庁（ＪＰ）　(12)**公開特許公報（A）**　(11)特許出願公開番号

特開平4－319734

(43)公開日　平成４年(1992)11月10日

(51)Int.Cl.[5]	識別記号	庁内整理番号	Ｆ１	技術表示箇所
Ｇ０６Ｆ　9/46	３４０	Ｆ　8120－5Ｂ		
12/00	５３５	Ｚ　8944－5Ｂ		
13/00	３５５	7368－5Ｂ		
15/16	３４０	Ａ　8840－5Ｌ		

審査請求　有　　請求項の数21（全 31 頁）

(21)出願番号　特願平4－16794

(22)出願日　平成４年(1992)１月31日

(31)優先権主張番号　６６０４５１

(32)優先日　1991年２月22日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　390009531
インターナショナル・ビジネス・マシーンズ・コーポレイション
INTERNATIONAL BUSINESS MASCHINES CORPORATION
アメリカ合衆国10504、ニューヨーク州　アーモンク　（番地なし）

(72)発明者　キヤスウオール・エイ・ハート
アメリカ合衆国10546、ニューヨーク州　ミルウツド、ルート　100、ポツクス　580

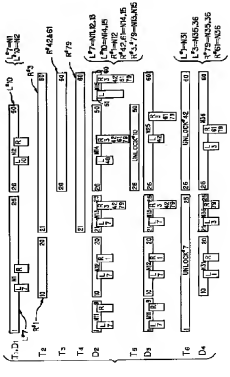
(74)代理人　弁理士　頓宮　孝一　（外３名）

(54)【発明の名称】　アドレス可能要素のロツク範囲のロツク状態を管理するための装置及び方法

(57)【要約】

【目的】　共用リソース中のアドレス可能要素の範囲をロックする互いに重複する複数のロック要求に対して効率的に対処し得る方法及び装置を提供する。

【構成】　ロックＬ＃７、１０が存在する既存のロック済範囲であるノードＮ１、Ｎ２に対して、該範囲と重複する新たなロック要求Ｒ＃１、３、４２、６１、７９を受け取ると、分割を行なってノードＮ１１～１４を生成し、更にノードＮ１５を生成する。この時、ノードＮ１１～１３のロックリストＬは（７）、ノードＮ１４のロックリストは（１０）のままであってＬ＃７、１０はそのまま保持されるが、ノードＮ１２～１５のレクエスタリストＲにはそれぞれ（１）、（３、４２、７９）、（３、４２、６１、７９）、（３、４２、６１、７９）が登録される。ロック解除要求ＵＮＬＯＣＫ＃１０を受け取ると、ノードＮ１４、１５の再結合が実行されてノードＮ２５が形成され、該ノードに対してＬ＃４２（Ｌ＃７と非重複）が生成される。同様にＵＮＬＯＣＫ＃７、４２が要求された時点でノードの再結合が実行され、重複しないＬ＃１、３が生成される。



1

【特許請求の範囲】

【請求項1】　互いにオーバラップする可能性のあるロック範囲であって、アドレス指定された要素の複数のロック範囲を表わすロック管理機構を含んでいる装置において、1つまたは複数のアドレス可能要素を表わす、アドレス可能要素の少なくとも1つの範囲の少なくとも1つの単独記述を生成及び維持する単独記述生成及び維持手段と、少なくとも1つのロック要求に応答して前記単独記述生成及び維持手段を制御し、前記アドレス可能要素の、互いにオーバラップしている複数の範囲の記述を分割して、互いにオーバラップしていない複数の範囲の単独記述にする制御手段とを備えたことを特徴とする装置。

【請求項2】　請求項1記載の装置において、前記制御手段は、ロック解除操作に応答して、前記アドレス可能要素の前記複数の範囲の単独記述を選択的に結合して、前記アドレス可能要素の少なくとも1つの範囲の少なくとも1つの別の単独記述を形成することを特徴とする装置。

【請求項3】　請求項1記載の装置において、前記単独記述の各々は、少なくとも1つのロック・オーナーによってその全体が保持されている単独記述か、或いは、前記ロック管理機構によって待ち状態を許可されたある1つのロック・リクエスタによってその全体が要求されている単独記述であり、且つ、アドレス可能要素の範囲の前記単独記述のうちの1つによって記述された前記範囲の各々は、既に要求がなされている範囲のうちの少なくとも1部分と、その範囲が同一であることを特徴とする装置。

【請求項4】　請求項1記載の装置において、該装置は更に、リクエスタに対する待ち状態の許可を阻止する手段であって、リクエスタが、第1の範囲に対して待ち状態にあるプロセスによってロックされている第2の範囲に対する待ち状態を既に許可されており、かつ、そのリクエスタが前記第1の範囲をロック状態に保持している場合に、待ち状態の許可を阻止する阻止手段を備えていることを特徴とする装置。

【請求項5】　アドレス可能要素の少なくとも1つのロック済範囲の表示を維持するためのロック管理機構を含んでいる装置において、前記ロック済範囲の前記表示を、前記アドレス可能要素の少なくとも1つの範囲の少なくとも1つの単独記述として維持する表示維持手段を備えたことを特徴とする装置。

【請求項6】　請求項5記載の装置において、前記表示維持手段は、単独記述分割手段を含んでおり、該単独記述分割手段は、ロック要求に応答して、少なくとも1つの前記単独記述を分割し、前記アドレス可能要素の、少なくとも2つの互いにオーバラップしない範囲の単独記述にする手段であることを特徴とする装置。

【請求項7】　請求項5記載の装置において、前記表示維持手段は、選択的単独記述結合手段を含んでおり、該選択的単独記述結合手段は、ロック解除操作に応答して、前記アドレス可能要素の、少なくとも2つの互いにオーバラップしておらず互いに隣接している範囲の単独記述を選択的に結合して、前記アドレス可能要素の1つの範囲の1つの単独記述にする手段であることを特徴とする装置。

2

【請求項8】　請求項5記載の装置において、前記単独記述の各々は、少なくとも1つのロック・オーナーによってその全体が保持されている単独記述か、或いは、前記ロック管理機構によって待ち状態を許可されたある1つのロック・リクエスタによってその全体が要求されている単独記述であり、且つ、アドレス可能要素の範囲の前記単独記述のうちの1つによって記述された前記範囲の各々は、既に要求がなされている範囲のうちの少なくとも1部分と、その範囲が同一であることを特徴とする装置。

【請求項9】　請求項5記載の装置において、該装置は更に、リクエスタに対する待ち状態の許可を阻止する手段であって、リクエスタが、第1の範囲に対して待ち状態にあるプロセスによってロックされている第2の範囲に対する待ち状態を既に許可されており、かつ、そのリクエスタが前記第1の範囲をロック状態に保持している場合に、待ち状態の許可を阻止する阻止手段を備えたことを特徴とする装置。

【請求項10】　アドレス可能要素の互いにオーバラップする複数の範囲のロック状態を管理するためのロック状態管理方法において、前記アドレス可能要素の範囲の少なくとも1つの単独記述を生成及び維持するステップを含んでおり、前記アドレス可能要素は、連続した一連のアドレスを有する前記単独記述の各々に対応しており、前記アドレス可能要素の前記ロック状態の全てが、前記単独記述のうちの1つによって表されるようにしたことを特徴とするロック状態管理方法。

【請求項11】　共用リソースのアドレス可能要素のロックを求めるロック要求を管理する方法において、共用リソースの要求された範囲に対するロックを許可するステップであって、そのロックの範囲が、ユーザがそのロックを要求した共用リソースの範囲と同一の範囲であるようにするロック許可ステップと、前記ロックの範囲の少なくとも一部分の単独記述を生成及び記憶する単独記述生成及び記憶ステップと、前記ロックと、別のユーザから発せられた共用リソースのある範囲のロックを求める更なるロック要求と、についての記述を生成する記述生成ステップであって、a）前記ロックの分割記述を生成及び記憶し、該ロックの該分割記述の一部分が、待ち状態が許可されている前記更なる要求の一部分と同一の範囲を持った前記アドレス可能要素の範囲の単独記述であるようにするロック分割記述生成及び記憶ステップ、b）前記更なる要求の分割記述を生成及び記憶し、該更

なる要求の該分割記述の一部分が、待ち状態が許可されている前記ロックの一部分と同一の範囲を持った前記アドレス可能要素の範囲の単独記述であるようにする要求分割記述生成及び記憶ステップ、ｃ）前記ロックの前記単独記述の範囲内で、前記更なる要求と同一の範囲を持った、１つの範囲についての待ち状態を生成及び記憶する待ち状態生成及び記憶ステップのうちの少なくとも１つのステップを含んでいる記述生成ステップとを含んでいることを特徴とするロック要求を管理する方法。

【請求項１２】　請求項１１記載の方法において、該方法は更に、前記ロックの前記分割記述と前記要求の前記分割記述との少なくとも一方の、互いに隣接した２つの範囲から、前記ロックと前記要求との少なくとも一方の範囲内での、１つの範囲の１つの単独記述を生成するステップであって、互いに隣接した２つの範囲の、夫々の範囲について待ち状態が許可されているリクエスタを示した夫々のリクエスタ・リストが、互いに同一のリストであるか、または、双方が共にヌル・リストである場合に、その１つの範囲の単独記述の生成を行なうようにした単独記述生成ステップ、を含んでいることを特徴とするロック要求を管理する方法。

【請求項１３】　請求項１１記載の方法において、該方法は更に、前記分割記述をサーチして、前記共用リソースのユーザによってロックされている範囲についての許可されている待ち状態の有無を調べるサーチ・ステップと、前記サーチ・ステップの実行中に、共用リソースのユーザによってロックされている範囲についての許可されている待ち状態が発見された場合に、待ち状態の許可を阻止する待ち状態許可阻止ステップとを含んでいることを特徴とするロック要求を管理する方法。

【請求項１４】　ロック管理機構において、共有リソース中のアドレス可能要素の複数の範囲について許可ないしは要求された複数のロックを識別するデータを包含するデータ構造を含んでおり、ロックとロックを求める要求との両方が存在しているときには、前記データ構造が、少なくとも全体がロック状態に保持されている範囲ないしは全体が要求されている範囲である、アドレス可能要素の連続したアドレス範囲を定義している単独記述のみから成るようにしたことを特徴とするロック管理機構。

【請求項１５】　請求項１４記載のロック管理機構において、前記データ構造はツリー構造であり、前記単独記述の各々は該ツリー構造のノードであることを特徴とするロック管理機構。

【請求項１６】　請求項１４記載のロック管理機構において、該機構は更に、複数の許可されているロック範囲のうちの少なくとも１つのロック範囲とオーバラップする範囲についてのロックを求めるロック要求と、前記単独記述のうちの１つによって記述されている少なくとも１つの他の範囲に隣接しているある１つの範囲のロックを解除するロック解除操作との、少なくとも一方に応答して、前記単独記述によって表示される前記範囲の大きさを動的に変化させる手段を含んでいることを特徴とするロック管理機構。

【請求項１７】　マルチ・タスク方式ないしマルチ・プロセッサ方式のデータ処理システムにおける共用リソースのアドレス可能要素のある範囲をロックするための方法であって、前記データ処理システム中のデータ及びファイルのある範囲をロックすることを求める、互いにそのロック範囲がオーバラップする複数のロック要求を、適切に処理するための方法において、前記ロック要求を動的に分割して、アドレス可能要素の範囲の単独記述を有する互いにオーバラップしない複数のセグメントにする分割ステップと、その分割の後に、分割してできたロック要求範囲に対して個別に許可を与える許可ステップとを含んでいることを特徴とする方法。

【請求項１８】　マルチ・タスク方式ないしマルチ・プロセッサ方式のデータ処理システムにおける共用リソースのアドレス可能要素のある範囲をロックするための装置であって、前記データ処理システム中のデータ及びファイルのある範囲をロックすることを求める、互いにそのロック範囲がオーバラップする複数のロック要求を、適切に処理するための装置において、前記ロック要求を動的に分割して、アドレス可能要素の範囲の単独記述を有する互いにオーバラップしない複数のセグメントにする分割手段と、その分割の後に、分割してできたロック要求範囲に対して個別に許可を与える許可手段とを備えたことを特徴とする装置。

【請求項１９】　複数のアドレス可能要素を管理するためのアドレス可能要素管理機構であって、前記複数のアドレス可能要素は、その各々が、少なくとも２種類のオーナーシップ状態を取り得るものであり、それらオーナーシップ状態のうちの一方は被所有状態であって、この状態においてはその被所有状態に対応した少なくとも１つのオーナーが特定され、更に前記複数のアドレス可能要素は、それらを複数のユーザが共用できるようにしてあり、それら複数のユーザの各々が、前記複数のアドレス可能要素の１つないし複数の範囲についての前記被所有状態を求める要求を発することができるようにしてあるアドレス可能要素管理機構において、前記複数のアドレス可能要素を動的に区分して、それら複数のアドレス可能要素の、互いにオーバラップしない複数のサブ範囲にするアドレス可能要素区分手段と、前記サブ範囲の各々に関連したオーナーのリストを生成及び維持するオーナー・リスト生成及び維持手段であって、該オーナー・リスト中のオーナーの各々が、当該サブ範囲中の前記アドレス可能要素の各々に関する前記被所有状態を有するものである、オーナー・リスト生成及び維持手段と、前記サブ範囲の各々に関連したウェイタのリストを生成及び維持するウェイタ・リスト生成及び維持手段であっ

て、該ウェイタ・リストの中のウェイタの各々が、少なくとも1つのサブ範囲の中の前記アドレス可能要素の各々に関する被所有状態の解除を待つものであり、ウェイタ・リスト生成及び維持手段と、任意の前記オーナー・リスト中の各々のオーナーをその他のオーナー・リスト中の同一のオーナーとリンクさせるオーナー・リンク操作手段であって、このリンク操作によって、前記ロック要求のうちの、前記同一のオーナーに対して許可されている前記被所有状態の範囲のロックを求める1つのロック要求に対応したリンク済リストを形成するオーナー・リンク操作手段と、任意の前記ウェイタ・リスト中の各々のウェイタをその他のウェイタ・リスト中の同一のウェイタとリンクさせるウェイタ・リンク操作手段であって、このリンク操作によって、前記ロック要求のうちの、前記同一のウェイタに対して許可されている前記被所有状態の範囲のロックを求める1つのロック要求に対応したリンク済リストを形成するウェイタ・リンク操作手段と、少なくとも1つの前記サブ範囲の中に包含されるアドレス可能要素を含んでいる少なくとも1つの範囲の複数のアドレス可能要素の被所有状態を求める、リクエスタからの更なる要求に応答して、少なくとも1つの前記サブ範囲を動的に更に区分して、前記少なくとも1つのサブ範囲に含まれる複数のアドレス可能要素のうちの、前記更なる要求の中に含まれるアドレス可能要素を、該少なくとも1つのサブ範囲に含まれる複数のアドレス可能要素のうちの、前記更なる要求の中に含まれないアドレス可能要素から分離するサブ範囲区分手段であって、前記複数のアドレス可能要素に対する区分と前記サブ範囲に対する区分との少なくとも一方を実行した結果として形成される少なくとも1つのサブ範囲の、前記オーナー・リストと前記ウェイタ・リストとの一方に、前記リクエスタを追加する手段を含んでいるサブ範囲区分手段とを備えたことを特徴とするアドレス可能要素管理機構。

【請求項20】　請求項19記載のアドレス可能要素管理機構において、該機構は更に、前記更なる要求と前記サブ範囲のうちの少なくとも1つのサブ範囲との間のオーバラップを検出するオーバラップ検出手段を備えており、前記サブ範囲区分手段は、該オーバラップ検出手段に応答するようにしてあることを特徴とするアドレス可能要素管理機構。

【請求項21】　請求項19記載のアドレス可能要素管理機構において、該機構は更に、指定されたリクエスタに関する前記オーナーシップ状態の解除を求める解除要求に応答して、該リクエスタを包含している所定のオーナー・リストから該リクエスタを削除するリクエスタ削除手段と、前記指定されたリクエスタの削除に応答して、アドレス可能要素の互いに隣接する複数のサブ範囲の間で、それらサブ範囲のオーナー・リスト及びウェイタ・リストを互いに比較するリスト比較手段と、前記リスト比較手段に応答して、同一のオーナー・リスト及びウェイタ・リストを有する互いに隣接するサブ範囲どうしを動的に再結合するサブ範囲再結合手段とを備えていることを特徴とするアドレス可能要素管理機構。

【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、広くは、マルチ・タスク方式ないしマルチ・プロセッサ方式のデータ処理システムの共用リソースの管理に関するものであり、より詳しくは、ロック範囲の管理方式に関するものである。例えば周辺機器やファイルないしメモリのバイト空間等の共用リソース中の、アドレス対象の要素をロックする場合に、複数のロック要求の要求範囲が互いにオーバラップする可能性があるが、そのようなときにも、本発明のこの管理方式によれば、それらアドレスされる要素のロック範囲を効率的にモニタ及び管理することができる。

【0002】

【従来の技術】マルチ・タスク方式ないしマルチ・プロセッサ方式のデータ処理システムにおいては、ファイル、テーブル、メモリ等のリソースを、非同期的に、ないしは同時に共用するということが一般的に行なわれている。ところで、多くの共用リソースは、それを、統合した1つの主体（エンティティ）として構成することも、或いは、複数の主体の集合として構成することもでき、複数の主体の集合として構成した場合、それら複数の主体の各々を、アドレス（指定）可能要素と呼んでいる。このように複数のアドレス可能要素として構成した共用リソースは、その利用の仕方という点に関して大きな特徴を持っており、それは、そのリソース中のそれらアドレス可能要素のうちの一部分を、ランダムにアクセスして、共用することができるということである。複数のアドレス可能要素のうちの一部分のことを、一般的には「範囲（range）」と呼んでいる。この範囲を記述するには、オフセット（リソースの始点アドレスからのオフセットという意味である）と呼ばれる始点位置と、範囲の長さと呼ばれる、連続するアドレス可能要素の必要個数とを用いて記述するようにしている。その特別の場合として、ただ1個のアドレス可能要素から成る範囲もあり得るが、これは、オフセットと、長さ「1」とによって、記述することができる。共用リソースの種類（タイプ）には、幾つかの種類がある。例えば、並列処理要素即ちアレイを構成している夫々のプロセッサも共用リソースとなり、また、データベースの中のインデクスないしレコードや、ファイルないしメモリの中のバイト、直接アクセス記憶装置（DASD）中のエントリも共用リソースとなり、更には、ディスプレイやプリンタをはじめとする一連の入出力装置（I／O）等のハードウェアも共用リソースとなり得る。

【0003】以上のものを含んでいる一般的な環境において、複数のプログラムを、即ち複数のプロセスを、1

7

つまたは複数のプロセッサの上で走らせることがある。その場合、一般的には、それら複数のプロセスは、同じデータ処理システムの上を、互いに同時に、ただし互いに他のプロセスについては何も知ることなくランすることになる。そのため、それら複数のプロセスのうちの２つ以上のプロセスが、互いに同時に、同じ共用リソースへのアクセスを必要とすることも起こり得る。これはリソース競合と呼ばれている。このリソース競合を解消するための方策として、共用リソースへのアクセスをモニタして秩序付けるためのシステム制御プログラムを用意しておき、それら複数のプロセスを、このシステム制御プログラムの監視下に置くようにするということが行なわれている。この場合、その共用リソースの中に存在しているアドレス対象の要素の種類と、要求されるアクセスの種類とに応じて、然るべきルールを予め定めておき、同じ時刻にアクセスされている、ないしはアクセスを要求されている要素に対して、そのルールを適用するようにしなければならない。具体的な一例として、共用リソースがデータベースである場合について説明するならば、読み書きアクセスが要求されたときや、互いに異なったオペレーティング・システム（例えばＤＯＳ、ＵＮＩＸ、ＯＳ／２（これらはいずれも商標である）等）が、同時に使用されている状況においては、アクセスの種類を、排他的アクセスとすることが望ましく、一方、例えば読取りのみのアクセスが要求されたとき等のように、アクセスの種類を、共用アクセスとすることが望ましいこともある。データベースの分野では、これらのアクセスを、ロック（施錠）と呼んでいる。更に説明しておくと、アクセスの各々の種類（タイプ）を、またたいては、アクセスの各種類に適用するルールのセットの各々を、状態、ないしは状態情報と呼んでいる。従って、排他的アクセスと、共用アクセスとは、互いに異なった２種類のロック状態ということになる。

【０００４】システム制御プログラムには、例えば、ＵＮＩＸ（ＵＮＩＸは、ＡＴ＆Ｔベル研究所の商標である）や、ＩＢＭ社のＭＶＳ（ＭＶＳは、多重仮想記憶の頭文字を取ったものである）等をはじめとする、種々のオペレーティング・システムがある。これらのシステム制御プログラムによって提供されるファイルは、バイトのストリングの形で構成されており、それらファイルは、それを複数のプロセスが共用できるようにしてある。一般的には、プロセスが、ファイルを読み取ってそのファイルに変更を加える際には、それらの読取りと変更とを、そのファイルの一部分だけを対象として実行するに過ぎない。だとし、そのファイルの内容の完全性を維持するためには、２つのプロセスが、同時に同一のデータに対して、変更を加えることがないようにしておく必要がある。即ち、最低限でも、データの然るべき範囲を任意にロックすることによって、そのデータの排他的使用を可能にするような、何らかの機構を備えておく必要がある。複数のバイトから成るある範囲をロックすることを求めるロック要求の形式としては、例えば、ロックしようとしているそれらバイトのうちの、先頭のバイトと末尾のバイトとを指定した形式とすれば良い。また、そのロック要求の形式が、もし、オフセット（即ち先頭のバイト）と長さとから成る形式であったならば、計算によってその末尾のバイトを算出すれば良い。

8

【０００５】オペレーティング・システムに関する概論ともいうべき情報は、例えばハロルド・ローリン及びハーベイ・Ｍ・ディーテルの共著「オペレーティング・システム」なる文献（the text book by Harold Lorin and Harvey M. Deitel entitled Operating Systems, published by Addison-Wesley Publishing Co. (1981)）の第１２章に記載されている。また、ＵＮＩＸオペレーティング・システムについての詳細な情報は、例えばブリアン・Ｗ・カーニハム及びロブ・パイクの共著「ＵＮＩＸプログラミング環境」なる文献（the book by Brian W. Kernigham and Rob Pike entitled The UNIX Programming Environment, published by Prentice-Hall, Inc. (1984)）に記載されている。また、ＭＶＳオペレーティング・システムについての詳細な情報は、例えばハーベイ・Ｍ・ディーテル著「オペレーティング・システム入門」なる文献（the book by Harvey M. Dietel entitled An Introduction to Operating Systems, published Addison-Wesley Publishing Co. (1984)）の第２１章に記載されている。この最後のハーベイ・Ｍ・ディーテルの文献は、その第１８章において、ＵＮＩＸシステムについても説明している。

【０００６】ロック要求を処理するための方式として、過去において用いられていた方式には多くの種類のものがある。それらいずれの方式でも、アドレス可能要素のある範囲をロックする場合の最も困難な問題は、複数のロック要求の要求範囲が互いにオーバラップすることがあるという問題であった。これは、例えば、あるロック要求によって、第６番と第７番のアドレス可能要素がロック状態に保持されているときに、別のロック要求によって、第１番から第１０００番までのアドレス可能要素のロックが要求されるというような場合である。

【０００７】この問題を解決するための様々な方式が既に存在しているが、それらの方式はいずれも、その問題を生じるシステムが特定の種類のものであり、且つ、要求されるアクセスが特定の種類のものである場合にしか、この問題を効率的に解決することができない方式である。それら公知の方式には、例えば、以下のものがある。

１．ビット・マップ方式。この方式は、アドレス可能要素の全空間をカバーするフラグを使用して個々のロックの存在を表示すると共に、補助的システムを使用して、それらロックの夫々のオーナーをモニタ及び識別し、また更に必要とあらば、範囲がオーバラップするロック要

求を分解するものである（この方式は基本的に、ロック要求がオーバラップすることが全くないか、或いはオーバラップするロック要求がめったに発生しない、小規模の共用リソースに適した方式である）。

2．リンクト・リスト方式。この方式は、ロック済範囲と、そのロック済範囲のロック状態を保持しているオーナーとを表示した情報を、リストに包含するものである（この方式は基本的に、高度に会話方式化されていないシステムや、ロック要求が高度にランダム化されておらずしかもロック要求の数が多くない上に共用ファイルの数が少なく（例えば1万以下）、プロセスの数も少ない（例えば600以下）システム、或いは、応答時間が短いことがそれほど重要視されないシステム（何故ならば、複数のリンクト・リストをサーチして衝突の有無を調べる作業は、本来的に時間のかかる作業だからである）に適した方式である）。

3．アドレス可能要素個別ロック方式。この方式は、アドレス可能要素の各々を1つの主体（エンティティ）と見なし、個別にロック情報を維持するものである（この方式は基本的に、予期されるロック範囲が狭い範囲である場合に、適した方式である）。

4．ラウンディング・アップ方式。この方式は、アドレス可能要素個別ロック方式と類似した方式であるが、ただし、共用リソースを予め複数の範囲に分割しておき、それら複数の範囲の各々を1つの独立した主体として取り扱うようにするという点が、アドレス可能要素個別ロック方式とは異なっている。この方式は、アドレス可能要素個別ロック方式の基本的な考え方を、個別の要素より大きいロック範囲へと、拡張した方式ということができる。

【0008】

【発明が解決しようとする課題】以上に列挙した公知の方式は、特定の種類のシステムの特定の種類の用途にのみ適した方式に過ぎず、それら方式のうちの1つとして、一般化された方式、即ち、様々なシステムの様々な用途に対して効率的に適用できるものはない。それらいずれの方式にも、その方式に固有の非効率性が付随している。例えば、ビット・マップ方式やアドレス可能要素個別ロック方式には、ハードウエアに関する非効率性が付随しており、また、ラウンディング・アップ方式では、要求した範囲を超える、より広い範囲の共用リソースの部分をロックしてしまうため、共用リソースの利用率に関する非効率性が付随することになる。

【0009】従って本発明の目的は、データ処理システムのファイル中の、アドレス可能要素のある範囲をロックするための、効率的なアルゴリズムを提供することにある。本発明の更なる目的は、データ処理システムの共用リソース（例えばデータやアプリケーション・プログラム等）の、ある範囲をロックすることを求める、互いにオーバラップする複数のロック要求を、適切に検出し、且つ取り扱うことができる、非常に高速で動作する機構を提供することにある。本発明の更なる目的は、広範な用途に好適に使用することができると共に、選択可能なロック状態の種類数にも、その共用リソースが含み得るアドレス可能要素の個数にも、また、ロック要求が含み得るアドレス可能要素の範囲の大きさにも、殆ど制約されることのないロック要求の管理（例えば状態情報の管理）を行なえるようにすることにある。本発明の更なる目的は、ハードウエアに関して効率的であり、しかも、ロック要求の要求範囲を超える、共用リソースのより大きな範囲をロックする必要をなくして、共用リソースの利用率においても効率的なものとすることができるようにした、ロック管理システムを提供することにある。

【0010】

【課題を解決するための手段】本発明の以上の目的並びにその他の目的を達成するため、本発明は、その1つの局面においては、複数のロック要求を管理するための、ロック要求管理方法を提供する。この方法は、前記アドレス可能要素の範囲の少なくとも1つの単独記述を生成及び維持するステップを含んでおり、前記アドレス可能要素が、連続した一連のアドレスを有する前記単独記述の各々に対応しているようにし、前記アドレス可能要素の前記ロック状態の全てが、前記単独記述のうちの1つによって表示されるようにしたことを特徴とする方法である。

【0011】また本発明は、その別の1つの局面においては、マルチ・タスク方式ないしマルチ・プロセッサ方式のデータ処理システムの共用リソースの、アドレス可能要素のある範囲をロックするための方法及び装置を提供する。これら方法及び装置は、前記データ処理システムの中のデータ及びファイルの、ある範囲をロックすることを求める、互いにそのロック範囲がオーバラップする複数のロック要求を遂切に処理するための方法及び装置であって、前記ロック要求を動的に分割して、アドレス可能要素の範囲の単独記述を有する互いにオーバラップしない複数のセグメントにする分割ステップないし分割手段と、その分割の後に、分割してできたロック要求範囲に対して個別に許可を与える許可ステップないし許可手段とを含んでいることを特徴とする方法及び装置である。

【0012】また本発明は、その更に別の1つの局面においては、互いにオーバラップする可能性のある、アドレスされる要素の複数のロック範囲を表わすロック管理機構を含んでいる装置を提供する。この装置は、1つまたは複数のアドレス可能要素を表わす、アドレス可能要素の少なくとも1つの範囲の少なくとも1つの単独記述を生成及び維持する単独記述生成及び維持手段と、少なくとも1つのロック要求に応答して前記単独記述生成及び維持手段を制御し、前記アドレス可能要素の、互いに

オーバラップしている複数の範囲の記述を分割して、互いにオーバラップしていない複数の範囲の単独記述にする、制御手段とを備えたことを特徴とする装置である。

【0013】また本発明は、その更に別の1つの局面においては、アドレス可能要素の少なくとも1つのロック状態範囲の表示を維持するためのロック管理機構を含んでいる装置を提供する。この装置は、前記少なくとも1つのロック状態範囲の前記表示を、前記アドレス可能要素の少なくとも1つの範囲の少なくとも1つの単独記述として維持する表示維持手段を備えたことを特徴とする装置である。

【0014】また本発明は、その更に別の1つの局面においては、アドレス可能要素の互いにオーバラップする複数の範囲のロック状態を管理するためのロック状態管理方法を提供する。この方法は、前記アドレス可能要素の範囲の少なくとも1つの単独記述を生成及び維持するステップを含んでおり、前記アドレス可能要素が、連続した一連のアドレスを有する前記単独記述の各々に対応しているようにし、前記アドレス可能要素の前記ロック状態の全てが、前記単独記述のうちの1つによって表示されるようにしたことを特徴とする方法である。

【0015】また本発明は、その更に別の1つの局面においては、共用リソースのアドレス可能要素のロックを求めるロック要求を管理する方法を提供する。この方法は、共用リソースの、要求された範囲についてのロックを許可するステップであって、そのロックの範囲が、ユーザがそのロックを要求した該共用リソースの範囲と、同一の範囲であるようにする、ロック許可ステップと、前記ロックの範囲の少なくとも一部分の単独記述を生成及び記憶する単独記述生成及び記憶ステップと、前記ロックと、前記共用リソースの別のユーザから発せられた、該共用リソースのある範囲のロックを求める更なるロック要求とについての記述を生成する記述生成ステップであって、a）前記ロックの分割記述を生成及び記憶し、該ロックの該分割記述の一部分が、待ち状態が許可されている前記更なる要求の一部分と同一の範囲を持った、前記アドレス可能要素の範囲の単独記述であるようにするロック分割記述生成及び記憶ステップ、b）前記更なる要求の分割記述を生成及び記憶し、該更なる要求の該分割記述の一部分が、待ち状態が許可されている前記ロックの一部分と同一の範囲を持った、前記アドレス可能要素の範囲の単独記述であるようにする、要求分割記述生成及び記憶ステップ、c）前記ロックの前記単独記述の範囲内で、前記更なる要求と同一の範囲を持った、1つの範囲についての待ち状態を生成及び記憶する待ち状態生成及び記憶ステップ、のうちの少なくとも1つのステップを含んでいる記述生成ステップとを含んでいることを特徴とする方法である。

【0016】また本発明は、その更に別の1つの局面においては、ロック管理機構を提供する。このロック管理機構は、共有リソース中のアドレス可能要素の複数の範囲について許可されないしは要求された複数のロックを識別するデータを包含するデータ構造を含んでおり、ロックとロックを求める要求との両方が存在しているときには、前記データ構造が、少なくとも全体がロック状態に保持されている範囲ないしは全体が要求されている範囲である、アドレス可能要素の連続したアドレス範囲を定義している単独記述のみから成るようにしたことを特徴とするロック管理機構である。

【0017】また本発明は、その更に別の1つの局面においては、複数のアドレス可能要素を管理するためのアドレス可能要素管理機構であって、前記複数のアドレス可能要素は、その各々が、少なくとも2種類のオーナーシップ状態を取り得るものであり、それらオーナーシップ状態のうちの一方は被所有状態であって、この状態においてはその被所有状態に対応した少なくとも1つのオーナーが特定されるようにした、アドレス可能要素管理機構を提供する。このアドレス可能要素管理機構は、「前記複数のアドレス可能要素を動的に区分して、それら複数のアドレス可能要素の、互いにオーバラップしない複数のサブ範囲にするアドレス可能要素区分手段」と、「前記サブ範囲の各々に関連したオーナーのリストを生成及び維持するオーナー・リスト生成及び維持手段であって、該オーナー・リスト中のオーナーの各々は、当該サブ範囲の中の前記アドレス可能要素の各々に関する前記被所有状態を有するものである、オーナー・リスト生成及び維持手段」と、「前記サブ範囲の各々に関連したウェイタのリストを生成及び維持するウェイタ・リスト生成及び維持手段であって、該ウェイタ・リストの中のウェイタの各々は、少なくとも1つのサブ範囲の中の前記アドレス可能要素の各々に関する被所有状態の解除を待つものである、ウェイタ・リスト生成及び維持手段」と、「任意の前記オーナー・リスト中の各々のオーナーをその他のオーナー・リスト中の同一のオーナーとリンクさせるオーナー・リンク操作手段であって、このリンク操作によって、前記ロック要求のうちの、前記同一のオーナーに対して許可されている前記被所有状態の範囲のロックを求める1つのロック要求に対応したリンク済リストを形成する、オーナー・リンク操作手段」と、「任意の前記ウェイタ・リストの中の各々のウェイタをその他のウェイタ・リストの中の同一のウェイタとリンクさせるウェイタ・リンク操作手段であって、このリンク操作によって、前記ロック要求のうちの、前記同一のウェイタに対して許可されている前記被所有状態の範囲のロックを求める1つのロック要求に対応したリンク済リストを形成する、ウェイタ・リンク操作手段」と、「少なくとも1つの前記サブ範囲の中に包含されるアドレス可能要素を含んでいる少なくとも1つの範囲の複数のアドレス可能要素の被所有状態を求める、リクエスタからの更なる要求に応答して、少なくとも1つの前

記サブ範囲を動的に更に区分して、前記少なくとも1つのサブ範囲に含まれる複数のアドレス可能要素のうちの、前記更なる要求の中に含まれるアドレス可能要素を、該少なくとも1つのサブ範囲に含まれる複数のアドレス可能要素のうちの、前記更なる要求の中に含まれないアドレス可能要素から分離するサブ範囲区分手段であって、前記複数のアドレス可能要素に対する区分と前記サブ範囲において対する区分との少なくとも一方を実行した結果形成されるなくとも1つのサブ範囲の、前記オーナー・リストと前記ウェイタ・リストとの一方に前記リクエスタを追加する手段を含んでいるサブ範囲区分手段」とを備えたことを特徴とするアドレス可能要素管理機構である。本発明の以上の目的、局面及び利点、並びに更なる目的、局面及び利点は、以下に述べる、添付図面に即した本発明の好適実施例の詳細な説明を参照することにより、更に明瞭に理解することができよう。

【0018】

【実施例】添付図面中、図1は、ＩＢＭ社のＳ／３７０ＬＡＮ（ローカル・エリア・ネットワーク）ファイル・サーバ・モデル（IBM S/370 LAN file server model）をブロック図の形式で示したものである。同図のシステムは、メインフレーム・コンピュータ１０を含んでおり、このメインフレーム・コンピュータ１０は、ＩＢＭシステム３７０ファミリーに所属するコンピュータのうちの1つである。また、このメインフレーム・コンピュータ１０には、Ｓ／３７０制御プログラム１２を実装してあり、図中に示したように、このプログラム１２は、ＶＭあるいはＭＶＳのプログラムである。メインフレーム・コンピュータ１０には、複数のＤＡＳＤ（直接アクセス記憶装置）１４、１６を接続してある。それらＤＡＳＤは、一般的にはディスク装置であり、それらディスク装置に、共用データないし共用ファイルを記憶させ、そしてそれら共用データないし共用ファイルを、メインフレーム・コンピュータ１０が管理している。

【0019】メインフレーム・コンピュータ１０には更に、Ｓ／３７０ＬＡＮアダプタ１８を接続してあり、このＬＡＮアダプタ１８を、ローカル・エリア・ネットワーク（ＬＡＮ）２０に接続してある。図中には、ＬＡＮを1つしか示していないが、容易に理解されるように、このメインフレーム・コンピュータ１０には、複数のＬＡＮを接続することができる。図示したＬＡＮの形式は、1つの具体例としてＩＢＭ社のトークン・リング・ネットワーク形式としてあるが、これは、その他の形式、例えば、バス接続形式や、スター接続形式等であっても良い。図示の具体例では、ＬＡＮ２０には複数のインテリジェント・ワークステーションを接続してあり、それらワークステーションが共通のデータ並びに実行プログラムを共用するようにしてある。この具体例における、それらワークステーションの内訳は、2台のＤＯＳ（ディスク・オペレーティング・システム）ワークステーション２２、２４、４台のＯＳ／２（オペレーティング・システム・シリーズ２）ワークステーション２６、２８、３０、３２、それに、2台のＡＩＸ（ＡＩＸはＵＮＩＸ（登録商標）に基づいた、ＩＢＭ社のオペレーティング・システムである）ワークステーション３４、３６である。それらのうち、ワークステーション２２及び２４は、例えばＩＢＭ社のＰＣ／ＸＴパーソナル・コンピュータとすることができ、ワークステーション２６、２８、３０、及び３２は、例えばＩＢＭ社のＰＳ／２モデル３０・２８６パーソナル・コンピュータとすることができ、またワークステーション３４及び３６は、ＩＢＭ社のＲＳ６０００とすることができる。

【0020】メインフレーム・コンピュータ１０には、ＬＡＮ２０との間で、並びにこのＬＡＮ２０に接続されている様々なワークステーションとの間で、通信を適当に行なえるように、ＬＡＮサーバ・プラットフォーム３８を実装してあり、このプラットフォーム３８は、マルチ・タスク方式のワーク・ディスパッチャーである。このプラットフォーム３８は、ＬＡＮ通信アプリケーション４０と、ロック管理機構４２と、キャッシュ管理機構４４と、共通共用ＬＡＮファイル・サーバ４６とをサポートしている。接続されている複数台のワークステーションは、互いに異質な（ヘテロジーニアスな）ものであることもあり得る。互いに「異質な」ワークステーションであるというのは、互いに異なった種類のオペレーティング・システムで走っているという意味であり、この具体例に即していえば、複数台のワークステーションが、ＤＯＳ、ＯＳ／２、ＡＩＸという、３種類の異なったオペレーティング・システムでランしているのが、これに該当する。このため、ＬＡＮサーバ・プラットフォーム３８は、それらオペレーティング・システムの夫々に対応したプロトコル・コンバータ４８、５０、及び５２を含んでいる必要がある。それゆえ、本実施例では、ＤＯＳとＯＳ／２との、２種類のオペレーティング・システムのためには、夫々に、ＳＭＢプロトコル・コンバータ５０と５２とを備えており、また、ＡＩＸオペレーティング・システムのためには、ＮＦＳプロトコル・コンバータ４８を備えている（ＮＦＳはサン・マイクロシステム社の登録商標であり、ネットワーク・ファイル・システムの頭文字を取ったものである）。これらのコンバータは、ワークステーションから発せられた要求を、共用ＬＡＮファイル・サーバ・アプリケーションが処理することのできる、共通形式の要求へと、変換するものである。

【0021】ＬＡＮ２０上に存在している、全てのインテリジェント・ワークステーションは、ＬＡＮファイル・サーバへ向けて同時に要求を発することによって、互いに同時に共用データの夫々の部分を更新することができるようにしてあるが、このとき、それらインテリジェント・ワークステーションの各々は、その他のインテリ

15

ジェント・ワークステーションが何をしようとしているかは知らない。また、共用データのうちの、ある1つの部分を定義するには、複数のバイトないし複数のレコードから成る範囲（range）として定義するようにしている。共用アプリケーション（ここではファイル・サーバ・アプリケーション）は、データの完全性並びに一貫性を維持するために、同時トランザクションどうし間のインターロック（interlocking）を実行する必要がある。インターロックを実行する際の各トランザクションのシーケンスは次のとおりである。(1) 該当する範囲をロック状態にする。(2) 夫々のデータを読み取って更新する。(3) 更新したデータをＤＡＳＤに書き込む。(4) ロック状態を解除する。

【００２２】図2は、本発明の好適実施例に係る、ロック要求の二進ツリー構造を示したものであり、この二進ツリー構造は、アドレス可能要素のロック済範囲をトラッキングするためのものである。ロック及びロック要求を規定した情報を包含するデータ構造はツリー構造にすることが好ましく、そうすれば、その情報のサーチを効率的に行なうことができる。ただし、効率的なサーチという点を除けば、そのデータ構造を具体的にどのように構築するかは、本発明を実施する上で特に重要なものではない。図示のツリー構造の中の各ノードは、アドレス可能要素の特定の範囲を表示している。図2～図5において、各ノードが対応している範囲は、そのノードの右側の2つのブロックの中の数字によって示されている。更に各ノードには、そのノードの、左下に位置するノードと、右下に位置するノードとを指し示す、2つのポインタを設定することができるようにしてある。左下に位置するノードは、そのノードが表示している範囲の始点のアドレス可能要素と終点のアドレス可能要素とが共に、その上位ノードの範囲よりも小さい番号のアドレス可能要素であり、そのため上位ノードの範囲との間で、範囲がオーバラップすることはない。また、右下に位置するノードは、そのノードが表示している範囲の始点のアドレス可能要素と終点のアドレス可能要素とが共に、その上位ノードの範囲よりも大きい番号のアドレス可能要素であり、そのため上位ノードの範囲との間で、範囲がオーバラップすることはない。各ノードのこれら2つのポインタは、図中においてはそのノードの左側の2コマのブロックで示してある。それらブロックは空白のまま図示してあるが、それらに数字を書き込んでないのは、ポインタがアドレスを指定する形式は、基本的に、任意の指定形式とすることができるからである。尚、例えば図2の左側の枝に示されているように、夫々のノードが表示している範囲は、必ずしも互いに連続する範囲であるとは限らない。

【００２３】本発明は、基本的にはロック管理機構に関するものであって、特定のツリー構造に関するものではない。しかしながら、本発明を実施する上では、ＡＶＬ

16

ツリーを採用することが好ましいと考えられる。その理由は、ＡＶＬツリーでは動的にバランスの再調節を行なえるため、最適なサーチ効率が得られるからである。ただし、現在ある、いかなる種類のサーチ・アルゴリズムないしツリー方式を用いて本発明を実施しても良く、また、ツリー構造を管理して、そのツリー構造のバランスを維持することも、通常行なわれていることである。好適に使用することのできるツリー構造の代表的なものとしては、Ｂツリー、Ｂ⁺ツリー、それにＡＶＬツリー等のツリー構造を挙げることができる。本開示を完璧なものとするために、後の説明の中では、プロセスの途中でツリー構造のバランス再調節を（ただし必要に応じて）実行すべき箇所を、その都度指摘することにする。ツリー構造についての更に詳細な情報は、例えばＣ・Ｊ・デイト著「データベース・システム入門」（the text book by C. J. Date entitled An Introduction to Database Systems, Volume 1, third edition, published by Addison-Wesley Publishing Co. (1981)）の、特にその第2章を参照されたい。またそれと共に、例えばクルーズ著「データ構造及びプログラム設計」（Kruse, Data Structures and Program Design, 2nd Ed., Prentice-Hall (1984)）の、特にその第３４４頁のＡＶＬツリーについての説明等を参照されたい。

【００２４】ツリーの各ノードには、更にその他の情報も包含させておく必要がある。その情報の中には、少なくとも、共用リソースの当該範囲（即ち、そのノードに対応した範囲）をロックしたプロセスを特定した情報が含まれていなければならない。また、もし、そのロックしたプロセスとは別のあるプロセスが当該範囲を排他的に使用しようとして、待ち状態にある（即ち、現在のロックが解除されてみずからがその範囲をロックできるようになるのを待っている）場合には、その待ち状態にあるプロセスのリンクト・リストを示すポインタも、その情報の中に含まれている必要がある。本発明を構築する上で好適に利用できる情報については、後に、特に図２１を参照しつつ、更に詳細に開示する。本発明の基本を理解するためには、とりあえず次のことを知っておけば充分である。それは、ロックのオーナー（オーナーとは、当該範囲をロックしているプロセス等のことである）が存在している場合には、そのロックのオーナーが特定されていなければならないこと、それに、各ノードには、そのノードに関係した（即ち、そのノードが表わしている範囲に関係した）全てのリクエスタ（リクエスタとは、当該範囲のロックを求めるロック要求を発したプロセス等のことである）を載せたリストが維持されていなければならないということである。

【００２５】本発明は、コンパクトなデータ構造を可能にするという特質と、高速のサーチを可能とするという特質との、2つの特質を備えている。また本発明によれば、複数のロック要求がオーバラップする（即ち、それ

らロック要求が要求しているロック範囲が互いにオーバラップする）という問題は、それらロック要求を動的に分割して、互いにオーバラップしない複数のセグメントにし、その後に、それらセグメントに対して個別に（原子的に）許可を与えるようにすることによって解決している。更に、ロック済範囲についての制御情報を二進ツリー構造の中に入れることによって、互いにオーバラップするロック範囲を、高速でサーチして容易に検出することができるようにしている。この制御情報に関するデータ構造は、ロック要求の分割操作の進展に伴って拡大して行く性質を持ち、また、そのデータ構造に基づいて、互いにオーバラップする複数のロック要求を処理するようにしているため、二進ツリー構造とするのが特に適している。このように、本発明は基本的には、互いにオーバラップする複数のロック要求を好適に取り扱うことを目的としたものであるが、本発明のロック分割方式と、ツリー構造とを組み合わせるならば、それによって複数のロック要求をモニタするための強力で効率的な構成が得られるのである。

【0026】以下に説明する本発明の構築態様は、複数のロック要求をシリアルに受け取る構成としてあり、それら複数のロック要求は一般的に、それらのロック範囲が互いに異なったものである（ただし、必ずしも互いに異なっていなければならないわけではない）。これに関して特に述べておくと、本発明は、データ構造（例えばツリー構造等）の操作の仕方や、データ構造に対して同時に複数の操作を実行しなければならない可能性の有無や、複数の要求が同時に発生した場合にそれらをシリアル化するためのシリアル化機構の種類等によって、殆ど影響されることのないものである。

【0027】本発明は、ある認識の上に立って成されたものであり、その認識とは、ロック要求を管理するための、公知のロック管理方式を拡張しただけでは、広範な種々の用途に適用できるロック管理機構を得ることはできないということであり、その理由は以下のとおりである。

1．ビット・マップ方式の場合。ビット・マップそれ自体が、ロックのオーナーとリクエスタとに関する充分な情報を包含させることが不可能なものである。また、ロック状態を実際に管理できるようにするには、大量のメモリを消費すると共に、別設のシステムを必要とし、更には、たとえオーナー及びリクエスタの情報を包含させた補助的なデータ構造を追加したとしても、互いにオーバラップする複数のロック要求の管理を行なえるようにすることはできない。

2．リンク・リスト方式の場合。この方式も本来的に、互いにオーバラップする複数のロック要求の管理を行なえるものではなく、また更に本来的に、サーチの実行が困難で、その実行には多くの時間を必要とする。そのため、共用リソースの規模が拡大するにつれて、或いはユーザ数が増大するにつれて、その方式が大がかりなものとなる性質があり、更には、システムが会話式のものであったり、或いはロック要求が高度にランダム化されていたりその数が多い場合には、その方式が複雑になると共に、サーチの実行が更に困難になる性質がある。

3．アドレス可能要素個別ロック方式の場合。この方式を使用すると、大量のメモリを消費すると共に、その計算処理のためのオーバヘッドも相当に大きなものとなる。その理由は、1つのロック要求の範囲、ないし1つのロック済範囲に含まれている複数のアドレス可能要素の各々を、独立した1つの同時要求として取り扱わねばならないからである。

4．ラウンディング・アップ方式の場合。この方式では、実際には要求されていない部分までをも含めた範囲がロックされるため、偽競合が発生する。またそれに加えて、個別にロック可能な範囲の大きさを予め定めてしまうため、そのロック範囲の大きさを、一般的なあらゆる種類の用途に対して最適な大きさとすることができない。更には、この方式は、本来的に、ハードウエアと計算処理オーバヘッドとのいずれかにしわ寄せが行く一方で、偽競合も発生し易い。

【0028】アドレス可能要素個別ロック方式とラウンディング・アップ方式とは、いずれも本来的にロックの個数を増大させずにはおかないものである。また、アクセスが終了したときにはロックを確実に解除しておかねばならないが、万一ロック解除が誤って行なわれなかったならば、これらの方式ではそのために偽競合が更に多く生じる可能性がある。何故ならば、もしあるアクセスの終了時に、アドレスされた複数の要素のうちの1つに対するロック解除が行なわれなかったならば、その1つの要素を含むそれ以後の全てのロック要求に対して、アクセスが拒絶されることになるからである。

【0029】既述の如く、多状態のロック状態（即ち、複数種類のロック状態）を取り得るようにすると共に、異なった種類のアクセスが可能であるようにすることが望ましい場合があり、本発明の好適実施例は、そのような複数状態に対しても、対応可能なものである。図1に示すように、互いに異なったオペレーティング・システム（例えばマイクロシステム社のＮＦＳとマイクロソフトウエア社のＤＯＳ）を使用している別々のユーザが同時に存在している場合には、ビット・ストリームのフォーマットが異なるために、アクセスしたデータが適当に提供されないことがある。例えば、ユーザどうしが互いに「同質」（ホモジーニアス）であれば（「同質」とは、同種のオペレーティング・システムを使用しているという意味である）、適用するロック規則は共用アクセスとするのがおそらく適当であると判断され、一方、互いに「異質」（ヘテロジーニアス）な複数の要求（即ち、異種のオペレーティング・システムから発せられた要求）が存在していたり、必然的に排他的でなければな

らないアクセス状態（例えば読み書きアクセス等）が存在しているならば、そのことから判断されるのは、排他的アクセスが必要に違いないということと、もしそのアクセスを許可して良いのであれば、排他的アクセスを許可すべきであるということである。従って、ロック状態を複数状態にすると、それに応じて待ち状態の種類も増やさねばならない場合もあり、その待ち状態としては、例えば、あるロック済範囲が共用アクセス（例えば読出しのみのアクセス等）を許可されている場合でも、ある一群のユーザに対してはそのロック済範囲へのアクセスを規制することによって、共用状態の一部規制を行ない、そのロックの期間中、一部のリクエスタを待ち状態に置くという形の待ち状態が必要になる可能性もある。ただし以下の説明においては、理解を容易にするために、先ず最初に排他的ロック状態（即ち、読み書きのための排他的ロック状態）を利用する場合について説明し、その後に、本発明のその基本的な態様を、複数状態並びに複雑な待ち規則へと拡張したものについて説明することにする。

【００３０】本発明の重要な部分は、互いに要求範囲がオーバラップする複数のロック要求に関する部分である。互いにオーバラップする範囲の重なり合い方には幾通りかの種類の形があり、それらの形の各々は、要求されている夫々の範囲の位置と、それら範囲が要求された順序とに基づいて区別されるものである。オーバラップの最も典型的な形は、重なり合う２つの範囲の間で、それらの開始アドレスも互いに異なっており、終点アドレスも互いに異なっているという場合のオーバラップの形である。また例えば、先行するロック要求が第１２８番要素から第５１２番要素までの範囲のロックを要求するものであり、後続のロック要求が第０番要素から第２５６番要素までの範囲のロックを要求するものであった場合には、これを「左側オーバラップ」と呼ぶようにしており、そのわけは、この場合、後続の要求は、先行する要求の中に含まれているどの要素のアドレスよりも若いアドレスを持った要素に対するロック要求を含んでいるからである。一方、これとは逆に、先行する要求が第０番～第２５６番要素のロックを要求するものであり、後続の要求が第１２８番～第５１２番要素のロックを要求するものであった場合には、これを「右側オーバラップ」と呼ぶようにしており、後続の要求がより番号の大きいアドレスを持った要素の側にずれて、先行する要求に オーバラップしているからである。尚、以下の説明においては、以上の２つの形のオーバラップをまとめて、その他の形のオーバラップから区別するために、これらの形を「単純オーバラップ」と呼ぶことにする。

【００３１】以上とは異なった別のオーバラップの形として、重なり合う２つの要求範囲の間で、夫々の一方の端の要素アドレスが、互いに同一であるという場合がある。これは、例えば、先行する要求が第１２８番～第５１２番要素のロックを要求するものであり、後続の要求が第０番～第１２８番要素のロックを要求するものであった場合である。この場合のオーバラップは、右側の端のアドレスが同一の、左側オーバラップであって、これを、「境界共有左側オーバラップ」と呼ぶようにしている。

【００３２】互いにオーバラップするロック要求の重なり合い方には、更に、含有形と包含形とがある。これら含有形と包含形との間の相違は、重なり合う２つの要求範囲の、要求が発せられた順番と、範囲の大きさとにある。例えば、先行する要求が第０番～第５１２番要素のロックを要求するものであり、後続の要求が第０番～第２５６番要素のロックを要求するものであった場合には、これを「左側共有含有形オーバーラップ」と呼ぶようにしており、その理由は、後続の要求の要求範囲が、先行する要求の要求範囲の中に「含有」されるものであり、しかも、後続の要求の「左側」の境界アドレスが、先行する要求のものと一致しているからである。また、完全包含形というオーバラップがあり、これは、例えば、先行する要求が第０番～第２５６番要素のロックを要求しており、後続の要求が第６４番～第１２８番要素のロックを要求しているという場合である。含有形の逆の形が包含形である。即ち、包含形とは、後続の要求の要求範囲が、先行する要求の要求範囲よりも広く、先行する要求の要求範囲を「包含」する場合のオーバラップをいうものである。含有形の場合と同様に、この包含形にも、左側共有包含形、右側共有包含形、それに完全包含形の、３種類の形がある。

【００３３】本発明の基本的な概念は、複数のロック要求の求めるロック範囲が互いにオーバラップするという問題に対して、グラフ理論の要素を適用することにあり、これについては、後に図１２を参照しつつ更に詳細に説明する。ここでその概要を説明すると、オーバラップするロック要求が発見されたときには、ロック要求の分割を行なって、互いにオーバラップしない新たなロック要求を作り出し、それら新たなロック要求を個々に処理するようにしたものである。具体的な例を挙げるならば、例えば、あるロック要求によって既に第１０番～第２０番要素がロックされているときに、後続のロック要求が第１５番～第２５番要素をロックしようとしたときには、既にロック済の範囲と、後続のロック要求の範囲との双方を分割して、新たに３つの範囲を生成する。この分割の結果生成されるそれら３つの範囲は、夫々、第１０番～第１４番、第１５番～第２０番、それに第２１番～第２５番の要素の範囲である。第１０番～第２０番の範囲のロック状態を保持していた元のホルダー（ホルダーとは「保持者」の意味であり、即ち、ロック要求を発して許可されたため、現在ある範囲をロック状態に保持しているプロセス等のことであって、「オーナー」と同義である）は、この分割の後には、それによって生成

された３つの範囲（以後、分割して生成された範囲のことを、特にサブ範囲（subrange）ということがある）のうち、第１０～第１４番要素に対応したサブ範囲と、第１５番～第２０番要素に対応したサブ範囲とを、保持、即ち所有する（即ちロックする）ことになる。また、後続のロック要求を発した第２のリクエスタは、第１５番～第２０番要素に対応したサブ範囲と、第２１番～第２５番要素に対応したサブ範囲に関して待ち状態となり（即ち、それらの２つのサブ範囲のロックが解除されて、みずからがそれらをロックすることができるようになるのを待つ状態となり）、換言すれば、この第２のリクエスタ（＝要求を発したプロセス）は、待ちプロセスになる。この第２のリクエスタが、ロックを許可されて、待ちプロセスから第２のホルダーになり、それによって、それらサブ範囲を排他的に使用できるようになるのは、要求した全ての部分（即ち、元の要求範囲に対応した全てのサブ範囲）の各々が、個別に（原子的に）許可できる状態になってからである（即ち、それら全てのサブ範囲を同時に許可できる状態になってからである）。オーバラップが発生したときの、このノード分割の操作の具体的な一例を、図３に示してある。

【００３４】図３と図２とを比較すると、図２では、図中の最も左側に位置している「最末端ノード（リーフ・ノード）」は、第１０番～第２０番要素に対応したロック範囲を表示しており、一方、図３では、この最末端ノードを分割して２つのノードにしている。分割して得られた、その一方のノードは、第１５番～第２０番要素に対応したロック範囲を表わすと共に、その一方のノードの更に末端側に接続した２つの末端ノードを指し示す２つのポインタを包含したノードとなっており、また、分割して得られた他方のノードは、図中左側の最末端ノードであって、第１０番～第１４番要素に対応したロック範囲を表わすノードとなっている。

【００３５】図３はまた、後続のロック要求を分割して２つのノードにする分割操作を示したものでもある。この場合、分割して得られた２つのノードのうちの左側のノードは、第１５番～第２０番要素のサブ範囲に対応したノードであり、後続のロック要求を発したリクエスタは、現在、ロック状態にあるこのノードのウェイタとして、このノードに登録されている、一方、分割して得られた２つのノードのうちの右側のノードは、第２１番～第２５番要素に対応したノードである。この右側のノードは、「弱ノード」と呼ばれるものである。「弱ノード」とは、そのノードのリクエスタが、そのノードにウェイタとして登録されており、従ってオーナーを持たないノードのことである。ノードがオーナーを持たないことがあり得るのは、本発明においては、１つの要求を分割して得られた複数の部分（即ちサブ範囲）の全てが同時に使用可能（即ちロック可能）になるまで、その要求に関してはロックを許可しないようにしているからである。斯かるプロセスとすることによって、各々のサブ範囲の一元的な（ユニタリな）記述を維持できるようにしているのである。また更に、斯かるプロセスとすることによって、実際にはロックすべきでない範囲（即ち、実際には、被所有状態にある他のロック済範囲と共に使用されるわけでない範囲）がロックされてしまうという事態を回避している。従って、ある範囲が、あるリクエスタのロック要求によって生成された弱ノードに表示されている（従ってそのリクエスタは、現在はその範囲を使用することができないでいる）場合でも、もし、後続の別のリクエスタの要求範囲が、その弱ノードに表示されている範囲のみ（或いは、その一部分のみ）であるか、或いは、その弱ノードに表示されている範囲と、現在使用可能な別の何らかの範囲との組み合わせであった場合には、その後続の別のリクエスタが、その弱ノードに表示されている範囲を即座に使用することができる。本発明のこの特徴は特に重要なものであり、何故ならば、この特徴によって、共用リソースの利用率を可及的に向上させることが可能となっているからである。

【００３６】以上の具体例から分かるように、ロックと要求との双方を分割して（従って、既にロック済の範囲と新たなロック要求の要求範囲との双方を分割して）、互いにオーバラップしない複数の範囲（即ちサブ範囲）にした上で、それらサブ範囲の各々のロック状態を、所有されている状態（即ち、実際にロックされている状態）、待たれているだけの状態（即ち、弱ノードの状態）、そして、所有され且つ待たれている状態のうちの、いずれかの状態として表示することができるようにしてある。ロックないし要求をどのように分割したかについては、当該範囲のリクエスタないしオーナーが関知する必要はなく、なぜならば、ロック及び要求を分割して作った複数のサブ範囲は、例えばポインタ等によって、それらを常に互いにリンクさせているからである。

【００３７】図４は、先に第１０番～第２０番要素の範囲がロック状態となっていたところへ、後から第１番～第２５番要素の範囲のロック要求が発せられた結果、完全包含形のオーバラップ状態が発生した場合に実行される、ノードの分割操作を示したものである。この場合、ロックの方は分割されず、要求の方が３つに分割されて、第１番～第９番要素、第１０番～第２０番要素、及び第２１番～第２５番要素の、３つのサブ範囲にされる。この分割の操作に伴って新たに生成される２つの最末端ノード（第１番～第９番要素のノードと、第２１番～第２５番要素のノード）に対するロックの許可は延期され、また、現在ロックされた状態にある第１０番～第２０番要素のノードと併せて、これら３つのノードの全てに、待ち状態にあることが表示される。

【００３８】図５は、先に第１０番～第２０番要素の範囲がロック状態となっていたところへ、後から第１４番～第１６番要素の範囲のロック要求が発せられた結果、

完全合有形のオーバラップ状態が発生した場合に実行される、ノードの分割操作を示したものである。この場合、ロックが（即ちそれまでロックされていた範囲が）分割されて、第１０番～第１３番要素、第１４番～第１６番要素、及び第１７番～第２０番要素の、３つのサブ範囲にされる。これら３つのサブ範囲の全ては、分割以前に既にロック状態にあったため、分割後も同じオーナーにより所有された状態を継続するが、ただし、それら３つのサブ範囲を表わすノードのうち、後から発せられたロック要求に対応した１つのノードにだけは、待ち状態が表示される。

【００３９】図３～図５に示した、ノードを分割する際の具体的なプロセスについては、後に図７及び図８のフローチャートと、図１３のフローチャートとを参照しつつ、更に詳細に説明する。図２～図５に関する上の説明において、特に注意すべき本発明の重要な特徴は、夫々のノードが対応している範囲は、そのいずれも、１つの範囲の全体が所有されているか、１つの範囲の全体が要求されているか（即ち待たれているか）、或いは、１つの範囲の全体が所有され且つ要求されているかの、いずれかであるということである。従って、共有リソースの範囲のうち、実際に使用されている範囲以外は、ロックされた状態になることはなく、また、それらアドレス可能要素の範囲（即ち共有リソースの範囲）のうち、実際に待たれている範囲以外は、それに対応した弱ノードが存在することはない。そのため、ロック状態ないしは弱ロック状態（弱ロック状態とは、実際にはロックされていない、弱ノードに対応した範囲の待ち状態のことである）が存在している範囲を記述する記述子の個数は、最小限で済むようになっている。これは、互いにオーバラップしたロック要求の範囲を分割して、互いにオーバラップしない複数の範囲（サブ範囲）にするために必要な個数のノードしか生成しないようにしているからである。記述子の個数が最小限であるという状態は、ロック解除の際に実行する後述のノードの再結合の操作が行なわれるときにも維持される。従って、偽競合が発生するおそれがなく、ロック状態をモニタするために使用するリソースの量も最小限に抑えることができ、また、システムの計算処理用資源の使用量も最小限に抑えることができる。更に、共用リソースの範囲のうち、ロック状態が許可される範囲は、その範囲の全体が使用される範囲であり、弱ノードが割り当てられる（即ち待ち状態が許可される）範囲は、その範囲の全体が要求されている範囲であるため、共用リソースを最大限に利用することができるのである。

【００４０】互いにオーバラップした要求範囲を分割して、互いにオーバラップしない複数の範囲（サブ範囲）にする方法については、以上に示したとおりであるが、本発明を理解するためには更に、本発明のロック管理システムによれば、ロック及び要求それ自体が分割及び再結合されるということを認識することが重要である。そしてそれを認識するためには、本発明の動作によって、ロック済範囲並びに待ち範囲を、一元的な（ユニタリな）方式で、一義的に記述することが可能になっているということを理解することが重要である。この一元的な、そして一義的な記述のことを、以下の説明においては、ロック済範囲ないし待ち範囲の「単独記述（singular description）」と呼ぶことにする。このように呼ぶ理由は、１つには、この記述が、ロック済範囲ないし待ち範囲の唯一の記述として一元化されていることによるものであり、またもう１つには、この記述が、範囲、オーナーシップ、ロック状態、及びウェイ・タリストに関して一義的であることによるものである。各々の単独記述の基本的な特質は、その単独記述が、連続したアドレスを有する複数のアドレス可能要素の、或いは、共用リソース中で連続している複数のアドレス可能要素の、１つの範囲の記述であるということにあり（この範囲は、例えば、先頭の要素のアドレスである始点アドレス即ちオフセットと、長さとによって記述することができる）、また、それらのアドレス可能要素の範囲は、その範囲の位置と、オーナーシップ状態（例えば、ウェイタしか存在していない無オーナー状態、オーナーが単数でロック状態は排他的であり更に単数または複数のウェイタが存在している状態、同じくオーナーが単数であってロック状態は排他的であるがウェイタが存在していない状態、それに、複数のオーナーが存在している状態等がある。更には、この最後の、複数のオーナーが存在しているオーナーシップ状態においても、そのときのロック状態との関連で、ウェイタが存在している場合としていない場合とがある）と、ロック状態（例えば、排他的ロック状態、共用ロック状態、等々）と、ウェイタ・リストとにおいて、一義的な範囲である。本発明では、ある単独記述によって記述した範囲と、別の単独記述で記述した範囲とが、決してオーバラップしないようにしている。そして、この非オーバラップ状態を維持するために、ロックと、それにオーバラップするロック要求とを（即ち、既存のロック済範囲と、その範囲にオーバラップする新たな要求の範囲とを）、複数の単独記述へ分割するようにしているのである。また、この分割とは逆の操作として、各々が単独記述によって記述された２つの隣接する範囲が、その記述によって表わされた範囲についてのみ独自の値を持ち、それ以外の、オーナーシップ状態、ロック状態、及びウェイタ・リストについては互いに同一であったならば、それら２つの単独記述を結合して、１つの単独記述にするということも行なっている。

【００４１】もう１つ重要なことがあり、それは、アドレス可能要素の互いにオーバラップしない範囲を記述するための、上で説明した単独記述と、オーナーが見るロック済範囲の記述、ないしはリクエスタが見る要求範囲

の記述とを、明確に区別して認識しておかねばならないということである。即ち、要求の記述の場合には、その要求に対して結果的にロックが許可されるにせよ、されないにせよ、その記述の構成中には、ロックを要求するアドレス可能要素の１つないし複数の範囲と、所望のロック状態の種類（例えば排他的ロック状態を望むか、それとも非排他的ロック状態を望むか）とが、当然含まれていなければならない。これに対して、オーバラップ状態がどのようになっているか、ないしは、オーナーシップ状態がどのようになっているかは、その要求を発したリクエスタにしてみれば、それらの状態によってロックないし待ち状態がそのリクエスタに対して許可されるか否かが決まるということを別にすれば、全く関知しないことである。即ち、既に指摘したように、本発明のユーザは、要求範囲ないしロック済範囲の分割及び再結合の操作については全く関知する必要がなく、なぜならば、単独記述は、それを見て利用するのはロック管理機構だけだからである。従って、各々の要求が発せられるたびに、多くのサブ範囲が生成される可能性があるが、ロック管理機構はそれらサブ範囲の各々の単独記述を維持して行くばかりでなく、それと共に、元の要求範囲が複数のサブ範囲に分割された後にも、元の要求の全体の記述を維持するようにしている。この元の要求の全体を維持する方法としては、元の要求が分割されて生成された複数のサブ範囲どうしを、水平にリンクさせるという方法を用いている。これによって、各サブ範囲の単独記述を維持すると同時に、リクエスタの目から見た、元の要求の形を維持できるようにしているのである。

【００４２】二進ツリー構造は、データをその構造とすることによって、そのデータのサーチが高速で行なえるようになるものであることが、広く認識されている。高速サーチが可能である理由は、理想的にバランスしたツリーにおいては、サーチの中の各々の照会動作ごとに、その調査対象のサブツリーのノードのうちの半数を、以後のサーチから除外して行けるからである。このため、本発明の好適実施例では、ＡＶＬ二進ツリー構造（このツリー構造自体は公知のものである）を採用しており、特にＡＶＬ二進ツリーとしたのは、このツリー構造は、動的なバランスの再調節が行なえるからであり、以下の本発明の説明は、この好適実施例に即して進めて行く。ただし容易に理解されるように、本発明には、サーチが容易な構造の二進ツリー構造であれば、どのような種類ものでも採用することができ、例えば「Ｂツリー構造」、「Ｂ**ツリー構造」、或いは、スキップ・リスト、等々も使用可能である。この二進ツリー構造は、メインフレーム・コンピュータ１０のメモリの中に構築してあり、それをロック管理機構４２が利用することによって、全範囲の中の、該当する夫々のアドレス可能要素の範囲の状態を、高速で調べることができるようにしている。もし、ある範囲が、現在、ロック済状態にあるならば、そのロック済範囲に関する情報は、このツリー構造のノードの中に保持されている。この情報は、通常はオーナー情報を含んでおり、また場合によって、待ち状態にあるプロセス（即ちウェイタ）のリストも含んでいる。

【００４３】ここで、待に次のことを述べておく。即ち、ノードは、ウェイタが幾つ存在していても、存在している全てのウェイタを特定した情報を記憶しておけるように構成してあるが、実際に判明したところによれば、本発明を採用すれば、それによって共用リソースを効率的に利用できるようになるため、所与の時刻に、ある１つの範囲に関してウェイタが存在している確率は、極めて小さなものになる。従って、本発明の効果によって、いかなるロック要求に対しても、殆どの場合、最小限の時間内にロックの許可が与えられるようになっており、それによって、ユーザにとっての利便性が向上していると共に、リソースを好適に共用できる最大許容ユーザ数も増大している。

【００４４】更に、包含形オーバラップが発生した結果として、ロック要求の分割が行なわれた場合には、たとえ、分割されたその部分（即ちサブ範囲）のうちに即刻使用可能な部分があっても、その範囲を直ちにロック状態にはしないことが肝要である。そのような範囲を直ちにロックしたならば、不慮のデッドロックが発生するおそれがあるからである。これは、アドレス可能要素の範囲のうちには、待ちプロセスだけが存在していて、ホルダー、即ちオーナーが存在していない範囲があるということに関係している。既に述べたように、この状態を「弱ロック状態」と呼び、この状態にあるノードを「弱ノード」と呼んでいる。また、ロック要求のオーバラップが発生したために、個々のロック要求を複数の部分に分割する際には、個々のロック要求の全体に関する情報も維持しておかねばならない。この情報の維持は、分割して得られた夫々の部分の単独記述を、水平にリンクさせることによって行なうようにしている。更に、全てのアプリケーションにおいて必要とされるわけではないが、一部のアプリケーションでは、ロック要求を発した各リクエスタごとに、そのリクエスタの複数のロック要求を連鎖させて、１つにまとめておくことが有用な場合がある。なぜ有用かといえば、そうしておけば、ある１つのリクエスタが障害を発生したときに、そのリクエスタに関する全てのロックを速やかに識別してロック解除することができ、それによって、データの回復を促進し、或いはデータの破壊を防止することができるからである。

【００４５】これより本発明を、その具体的な実施例に即して、また特に、図６～図１５のフローチャートを参照しつつ、更に詳細に説明して行く。

【００４６】図６は、本発明の基本的な動作プロセスを示したものである。この図６、並びにそれに続く図７～

図１３では、サブルーチンによって構成されている処理ブロックは、左右両辺を二重線にした長方形のブロックで表わしてある。ロック状態の変更を求めるプロセスが起動されたならば、先ずブロック１０１で、初期化が必要か否かを判断する。初期化が必要と判断されたならば、ブロック１０２において、ノード情報を記憶させておくためのノード記憶空間が規定されているか否かを調べる。それが規定されていなかったならば、サブルーチン（ブロック１０３）を走らせて、その記憶空間を割り当てさせる。一方、その記憶空間が既に先取り（prefetch）されていたならば、この初期化処理手順では、続いてブロック１０４において、その記憶空間の中に、フリー・ノード・リスト（これは、利用可能なノードのリストであって、「フリー・ノード・プール」、或いは単に「ノード・プール」ともいう）を作成する。ブロック１０５では、そのリスト、即ちノード・プールの中のノードの個数を記憶すると共に、各々のノードの先頭位置を指し示すポインタを設定する。この後、初期化処理の完了を知らせる信号を送出して、この初期化処理手順は終了する。

【００４７】この初期化処理が完了した後に、ロック状態の変更を求める要求が発生した場合には、ブロック１０１における判断の結果、その「ＮＯ」経路をたどってブロック１０７へ進み、このブロック１０７では、そのロック状態の変更要求が、有効なロック要求か否かを判断する。もし、それが有効なロック要求であると判断されたならば、範囲ロック処理サブルーチン（ブロック１０８；このサブルーチンは図７及び図８に詳細に示してある）を起動する。一方、そのロック状態の変更要求が、有効なロック要求ではないと判断されたならば、ブロック１０９において、その要求が有効なロック解除要求か否かを判断し、もしそうであったならば、範囲ロック解除処理サブルーチン（ブロック１１０；このサブルーチンは図９及び図１０に詳細に示してある）を実行する。

【００４８】解除強制要求というのは、現在存在しているロック状態の中断を強制する要求である。この解除強制要求は、本発明の基礎概念という観点からは、特に重要なものではないが、ただし、この要求を利用できるようにしておくと役立つことが多い。そこで、本実施例では、この特徴を装備するために、ブロック１１１において更に判断を行ない、有効な解除強制要求の検出を行なうようにしている。もし、ブロック１０７、ブロック１０９、及びブロック１１１の夫々において、有効なロック要求、有効なロック解除要求、それに有効な解除強制要求が検出されず、そのため、それらいずれのブロックからも「ＹＥＳ」経路へ分岐しなかった場合には、このルーチンは、ブロック１１４において、「無効範囲要求」信号を返して処理を終了する。一方、有効な解除強制要求が検出されたならば、このルーチンは、ブロック１１２において、そのユーザの（即ち、その要求を発したリクエスタの）ロック済範囲キューを調べて、そのキューの中の先頭のロック・ノードを取り出す。ブロック１１３では、このユーザのロック済範囲キューを調査して、ロック済範囲が実際に存在しているか否か（即ち、そのロック済範囲キューの中から実際にロック・ノードを取り出せたか否か）を調べ、実際にロック済範囲が存在しているときが分かったならば、範囲ロック解除処理サブルーチン（ブロック１１０；図９及び１０に詳細に示す）を起動して、各々のロック・ノードに関してロック解除を実行し、そして更に、そのキューの中のロック・ノードを次々と取り出してロック解除を行なって行く。そして、このキューが空になったならば、ブロック１１５において、解除強制処理が完了したことを示すリターン信号（解除強制処理完了信号）を返送する。

【００４９】ブロック１０７において、有効なロック要求が検出された場合には、図７及び図８に示した範囲ロック処理サブルーチンが起動される。このサブルーチンにおける処理は、以下のとおりである。先ずブロック２０１で、ロックすべき範囲を表示したデータを取り出し、ブロック２０２で、ロック・ノードを取り出してそれを初期化し、ブロック２０３では、その初期化したロック・ノードを、ユーザの所有のロック済範囲のキューの中へ入れる。更にブロック２０４では、そのロック・ノードをロック・ツリーの中へ入れる。続いてブロック２０５では、そのロック・ツリーをサーチして、ロック・ノードどうしの間に衝突が存在しているか否かを判断する。このブロック２０５の衝突検出に関して重要なことは、ここで衝突が検出されたならば、更に図１３の分割プロセスによって新たなノードが生成され、そしてその新たなノードが、ロック・ツリーに付加されるということである。即ち、分割プロセスの実行中には、新たに生成したノードをスタックに入れてキューを形成し、その後に、その新たなノードを、そのキューから抜き出してロック・ツリーへ付加し、そして更に、衝突の有無を調べるようにしている。この分割のプロセスは、全ての衝突が消滅するまで、反復して実行しなければならない。本発明のこの点についての理解を容易にするには、図７の中の、一点鎖線２４０で囲んだ部分を１つのまとまった分割ステップであると考えるようにし、そして、その分割ステップの中の各々のサブステップが、図１３のプロセスに対応したものと考えれば良い。この場合には、図８の、判断ブロック２２２～２３０の各々において判断して得たデータを、図１３のブロック５０１において使用し、それによって、図１３の中の、左側ピアないし右側ピアの分割操作を制御するようにする（「ピア」の意味については後述する）。

【００５０】ロック要求のうちには、ロックを許可できない場合に、その要求に対して、待ち状態を許可するという応答を返して良いものと、待ち状態を許可せずに拒

29

絶応答を返さねばならないものとがある。そこで、いかなるロック要求にも、そのロック要求にとって、待ち状態が許容可能な応答のうちの1つであるか否かを表示したデータを包含させるようにしている。そして、ブロック205において、衝突が存在すると判断されたときには、ブロック206で最初のテストを行ない、そのロック要求を発したリクエスタを、待ち状態にしても良いのか、それとも、そのロック要求を即座に拒絶する必要があるのかを判断する。もし、待ち状態が許容不可であると判断されたならば、ブロック207で「範囲は既にロック済」信号を返送し、ブロック208でそのロック要求に関するノードを除去し、そしてブロック209において、その除去したノードをフリー・ノード・リストへ戻すようにする。

【0051】一方、そのロック要求が、待ち状態を許容するものであったならば、ブロック210において、そのロック要求の要求範囲が、いずれかのロック・ノードに表示されている、既存のある1つのロック済範囲と、完全に同一の範囲であるか否かを判断する。もし、そのロック要求の要求範囲が、既存のロック済範囲のうちの1つと完全に同一の範囲であれば、分割操作は不要であり、従ってその範囲を対象とした待ち状態の許可なり拒絶なりのプロセスを、即座に開始してかまわない。一方、既存のロック済範囲のうちに、そのロック要求の要求範囲と完全に同一の範囲のものがなかったならば、その要求範囲に対してグラフ理論の概念を適用するようにする。ここで適用するグラフ理論は、分割操作を制御するためのパラメータを得るための、その要求が衝突した既存のロック済範囲を対象とした一連のテストである。この分割操作は、図13に従って実行し、分割によって生成したノードはスタックに格納する。

【0052】続いて、待ち状態を許可するためのプロセスを開始する。この待ち状態許可プロセスでは、先ず、ブロック211において、その待ち状態を許可した場合にデッドロックが発生し得るか否かのチェックを行なう。このチェックのためのテストは、図14及び図15に示したサブルーチンに従って実行するのが好ましく、このサブルーチンについては後に詳述する。このチェックの結果、ブロック212において、デッドロックが発生し得ることが検出されたならば、ブロック214において、その事を示すリターン信号を発生した後に、図中に「D1」で経路を示したように、ブロック208、ブロック209を実行して、そのノードをフリー・ノード・リストへ戻してこのプロセスを完了する。一方、デッドロックが発生するおそれがないということが検出されたならば、ブロック213において、待ち状態を許可する。更なる分割が必要な場合には、そのことが、ブロック215で検出される。そして、それが検出されたならば、ブロック231で、分割スタックのキューから1つのノード（先頭のノード）を抜き出し、ブロック204

30

で、その抜き出したノードをロック・ツリーへ挿入し、更にブロック205で、衝突の有無を調べる。このプロセスは、全ての衝突が消滅するまで反復して実行する。全ての衝突が消滅したならばブロック221へ進み、そこでは、もし必要ならば、二進ツリーであるノード・ツリーのバランス再調節を実行して、効率的なサーチ性能を維持する。この後、ブロック215において分割スタックの最終チェックを行なう。このチェックによって、更なる分割の操作が不要であることが分かったならば、ブロック216において、ロック可能かそれとも待たねばならないかの状況に応じて、プロセスは分岐する。即ち、ロックが可能であれば「ロック許可」信号を返すようにし、一方、ユーザが待たねばならないのであれば、所定の待ち時間の「待ちサブルーチン」を実行する。このサブルーチンは、任意の公知の形態のもので良い。もしこの所定の待ち時間の間に、要求していた範囲のロックが解除されたならば、そのロック範囲をユーザに確承させて（ブロック219）、ブロック218で「ロック許可」信号を返送する。一方、その所定の時間内に、その範囲のロックが解除されなかった場合には、ブロック220で「ロック待ち時間切れ」信号を返送する。

【0053】図13は、ノード及び範囲を分割するための、ノード／範囲の分割操作を示したものである。この分割操作は以下のとおりである。先ずブロック501で、ブロック222～230でのテストの結果に基づいて条件を設定する と共に、分割点の位置を入力する。続いてブロック502で、サブルーチンを起動して、フリー・ノード・リストの中から新たなノード（第2のノード）を取り出し、ブロック503では、その新たなノードの中へ、分割しようとしている元のノード（第1ノード）のデータを複写する。ブロック504のステップとブロック505のステップとは、どちらを先にしても良く、これらによって、ブロック506以下のプロセス、或いはブロック507の以下プロセスへと分岐する。それらプロセスに含まれているステップは、記憶と単純な計算処理とだけであり、それらは、分割操作の結果生成される範囲（サブ範囲）の、始点と終点とを求めるためのものである。こうして得られたサブ範囲に関するデータと共に、ピアどうしを連結するリンク・データも併せて記憶するようにしており、そのデータ構造は、例えば図21に例示したデータ構造のようにする。尚、「ピア（peer）」とは、本来は仲間の意味であり、ここでは、1つのロック要求が分割された結果、生成された複数のサブ範囲どうしを、互いに「ピア」であるといい、また、分割によって生成された、1つのロック要求に関する複数のノード（それらノードは夫々にサブ範囲に対応している）どうしを、互いに「ピア」であるというように使用している。また、アドレス可能要素の番号が小さい方の範囲、ないしはその範囲に対応したノードを「左側ピア」といい、反対に、番号が大きい方の範囲ないし

31

ノードを「右側ピア」というようにしている。

【0054】図6に説明を戻し、そのブロック109においてロック解除要求が検出された場合には、ブロック110に進み、図9及び図10に示した範囲ロック解除サブルーチンが起動される。この範囲ロック解除サブルーチンは、このようにブロック109からの分岐によって起動されることもあれば、ブロック111からの分岐によって（ブロック112、113を介して）起動されることもあるため、それら2とおりの状況を区別して認識しておく必要があり、その識別をブロック301（図9）で行なっている。今この説明では、ロック解除の動作のために、このサブルーチンへ入ったものとしているのであるから、ここでは、処理はブロック302へ進み、そこで、ロック解除を実行すべき範囲を入力Pリストから取り出す。続いてブロック303で、ユーザが所有しているロックのキュー（所有ロック済範囲キュー）をサーチして、ブロック302で取り出したロック解除範囲に該当する範囲を有するノードを捜し出す。もし該当するノードが発見されなかったならば（ブロック304）、ブロック306において「ユーザによってロックされていない範囲」信号であるリターン信号を発生して、この処理手順を終了する。一方、ブロック304において、該当するノードが発見されたならば、ブロック305で、そのノードをユーザの所有ロック・キューから抜き取る。続いてブロック307において、そのノードが分割されたものであると仮定して、そのノードのピアのうちの、最も左側のピアを捜し出して、そこからロック解除操作を開始する。更に、ブロック308において、その最も左側のピアの右側に位置するピアを、スタックの中でプッシュし、プッシュされたピアは、その次にロック解除されることになる。

【0055】ブロック309では、最も左側のピアである上記ノードが、オーナー・ノードであるのか、それとも、弱ノード（即ちウェイタしか存在していないノード）であるのかを判断する。もし、そのノードがオーナー・ノードであったならば、ブロック330において、そのノードの範囲について待ち状態を許可されたウェイタが存在しているか否かを判断する。もし、そのノードに、ウェイタが存在していないことが示されていたならば、そのノードに対応した範囲は、その範囲のロックを求める要求が次に発生したときにロックを許可することができる。それゆえ、そのノードをツリーから除去し、必要とあらば、ツリーのバランス再調節を実行した上で（図10のブロック331）、そのノードをフリー・ノード・プールへ戻す（ブロック313）。一方、ブロック330において、そのノードの範囲について待ち状態にあるウェイタが存在していると判断されたならば、ブロック332において、そのうちの先頭のウェイタについて、それが時間切れを生じているか否かを調べる。もしそのウェイタが、時間切れであったならば、ブロック

32

336において、そのウェイタが既に時間切れ処理のためのスタックに入れられているか否かを調べ、ブロック337において、必要とあらばそのウェイタを（即ち、そのウェイタのノードを）時間切れスタックへ入れた上で、その旨を表示するフラグを立てる。続いて、ブロック338において、そのウェイタのノードをウェイタ・キューから抜き出し、ブロック330で再度ウェイタ・キューを調べ、そしてこのプロセスを、時間切れしていないウェイタを発見するか、或いは、ウェイタ・キューが空になるまで、反復して実行する。そして、ブロック332において、時間切れしていないウェイタを発見したならば、ブロック333においてそのウェイタをこの範囲の新たなオーナーにする。このブロック333では更に、この範囲のその新オーナーに関する「待ち数」をデクリメントし（この「待ち数」とは、ある1つのリクエスタが幾つの範囲においてウェイタとなっているかを表わす数である）、その後、ブロック334においてその待ち数を調べ、その新オーナーが待ち状態にある範囲が他に存在していない（すなわち、待ち数＝0）と判断されたならば、ブロック335において、この範囲を（即ち、この範囲に対応したノードを）ユーザの継承のために再結合キューへ入れる。一方、ブロック334において、その新オーナーが別のロック済範囲に対して待ち状態にあると判断されたならば、ロックを許可せずに、その新オーナーのノードを、フリー・ノード・プールへ戻す（図10のブロック313）。

【0056】ブロック309においてノードを調べたときに、それが弱ノードであったならば、即ち、ウェイタが存在しているだけのノードであったならば、ブロック310において、先頭のウェイタ（ウェイタが複数存在している場合には、それらはウェイタ・キューに入れられているため、そのキューの先頭のウェイタ）について、時間切れしているか否かを調べ、もし時間切れしていたならば、ブロック311で必要の有無を調べた上で、ブロック312において、その時間切れのウェイタのノードを、そのマスター・ノード（即ち、その時点において、その弱ノードが関係しているロックを制御しているノード）のキューから抜き取る。ただし、時間切れであってもなくても、そのノードは、ブロック313において、フリー・ノード・プール（利用可能ノード・プール）へ戻される。この後ブロック314において、そのノードのピアのうち、すぐ右隣のピアをロック解除スタックから取り出し、そして以上のプロセスをロック解除スタックが空になるまで、反復実行する。これによって、最終的には全てのピアのロック解除が行なわれ、しかも、その各々のピアの範囲に対して待ち状態にあるプロセスのリスト（即ち、各ピアのウェイタ・リスト）を考慮した適切なロック解除が行なわれることになる。

【0057】ブロック315において、ロック解除スタックが空になったと判別したならば、続いてブロック3

１７において時間切れスタックを調べ、時間切れスタックが空でなかったならば、ブロック３１８において、この時間切れスタックから先頭のノードを抜き出した上で、このプロセスをブロック３０７へ戻して、時間切れのために不要となった分割についての再結合の処理を行なう。一方、ブロック３１７において、時間切れスタックが空であったならば、ブロック３１９において、再結合スタックをチェックし、ブロック３２０において、そのスタックに内容が存在しているかまたは空であるかを調べ、再結合スタックが空でなかったならば、ブロック３２１において、再結合スタックからそのノードを抜き出し、そしてブロック３２２に移行して図１１及び図１２に示したフローチャートに従って、再結合の処理を実行する。尚、以上のプロセスは、再結合が全て完了するまで、即ち、再結合スタックが空になるまで反復実行される。

【００５８】以上に説明した図９及び図１０のプロセスは、解除強制要求が発せられて、図６のブロック１１１から分岐してこのプロセス（ブロック１１０）へ入ってきたとき（従ってそのことが図９のブロック３０１において識別されたとき）にも、同様の処理を行なうものであり、即ち、ブロック３０１での分岐によって、ユーザ・リストのサーチが省略されることが異なる以外は、全く同一の処理を行なうものである。このサーチが、解除強制の場合には不要であることは、いうまでもない。

【００５９】本発明は、その好適な実施態様としては、３種類の異なったロック解除動作を行なえるようにすることができる。上で説明したロック解除動作は、ロックのオーナーが単数であり、しかも単一の要求に応答して、１つまたは複数のノードをロック解除する動作であった。これに対して、特定の１つのオーナーから発せられた全ての要求に関係した全てのロックを解除できるようにしておけば、例えばそのオーナーに関する障害が検出されたとき等に、データの破壊を回避することができ、好都合である。また、これは基本的に解除強制要求の一変形であるが、ある１つの要求に関する全ての同時オーナーの全てのロックを解除できるようにしておくことも有用であり、そうしておけば、例えば、ロック状態を共用ロック状態から排他的ロック状態へ変更することができる。このような他の種類のロック解除動作は、以上に説明した処理手順を単に反復するだけでも容易に達成することができ、それによって所望のロック解除を行なうことができる。

【００６０】図１０のブロック３２２の再結合のプロセスを詳細に示したものが、図１１及び図１２のフローチャートである。ここで特に注意しておかねばならない重要なことは、この再結合のプロセスの動作を開始した時点では、待ち状態にあるロック要求に対応したノード（即ちウェイタ）に関する、全てのソート作業が既に完了しており、その結果、新たにまとめて所有されるようになる互いに隣接した範囲、即ち、互いに完全に同一のウェイタ・リストを持つ互いに隣接した範囲か、或いは、いずれも全くウェイタを持たない互いに隣接した範囲は、その全てが既に分かっているということである。そこで、先ず図１１のブロック４０１では、再結合処理の対象とすべきノード（これは、１つのグループを成す（即ち同一のロック要求から生成された）複数のピアのうちの、最も左側のピアである）に関して、そのピアにとっての「セントラル・ノード」を捜し出す。「セントラル・ノード」とは、１つのロック要求が複数のサブ範囲に分割されているときに、それら複数のサブ範囲の全てに関する大域的情報を包含しているノードであり、１つのロック要求に対して１つだけ存在している。もし、最も左側のピアのノード（以下、最左端ノードという）と、捜し出したセントラル・ノードとが同一のノードであったならば、そのセントラル・ノードには、それより右側のピアしか存在せず、また更に、もしそのセントラル・ノードには、ピアが１つも存在していないことをブロック４０２で判別すると、再結合の処理は不要である。即ち、再結合すべきものは存在しているため（ブロック４０３）、ブロック４０４においてユーザ／オーナを継承するためのサブルーチンをコールし、その後にブロック４０５において「ロック解除完了」信号を返送する。一方、そのセントラル・ノードにピアが存在していると、ブロック４０２で判定すると、先ずブロック４０６で、そのセントラル・ノードをチェックして、左側ピアの識別情報（ＩＤ）の有無を調べ、即ち、そのセントラル・ノードに左側ピアが存在しているか否かを調べる。左側ピアが存在していなかったならば、ブロック４０７において再びセントラル・ノードをチェックして右側ピアが存在しているか否かを調べる。もし右側ピアも存在していなければ、再結合は不要ということであり、そのためこのプロセスは、再結合が不要な場合の上述のシーケンスを実行して終了する。

【００６１】一方、そのセントラル・ノードに左側ピアが存在していたならば、ブロック４０８において更にそのセントラル・ノードを調査して、そのセントラル・ノードにウェイタが存在しているか否かを調べる。そして、そのセントラル・ノードにウェイタが存在していた場合も、存在していなかった場合も、ブロック４０９と４１３とのいずれかにおいて、そのセントラル・ノードのすぐ左隣に位置している左側ピアに、ウェイタが存在しているか否かを調べる。そのセントラル・ノードとその左側ピアのノードとのいずれにもウェイタが存在していなかったならば、ブロック４１０において、それら２つのノードを再結合し、かつブロック４１１において、それまで左側ノードであったノードをロック・ツリーから除去し、しかも必要とあらば、そのロック・ツリーのバランス再調整を行なった上で、ブロック４１２において、その抜き取ったノードを利用可能ノード・プールへ

戻す。一方、ブロック４０８、４１３において、そのセントラル・ノードにウェイタが存在しており、しかも、そのセントラル・ノードのすぐ左隣の左側ピアにもウェイタが存在していると判別したならば、ブロック４１５において、まだ他に左側ピアが存在しているか否かを調べ、そして更に、その「更なる左側のピア」にウェイタが存在しているか否かを調べる。このプロセスは反復実行されるが、セントラル・ノードの左側に、ウェイタが登録されていないピア（即ちノード）が発見されたならば、このループから脱出して、ブロック４１５において、まだ他に左側ピアが存在しているか否かのチェックを実行し、それが存在していたならば、ブロック４１６に移行してそのノードに関するウェイタの有無を調べ、ウェイタが発見されたならば、このプロセスは、再びブロック４１４へ戻る。基本的に、このプロセスは、このように反復されることによって左右を見渡し、ウェイタが存在していないか、或いは、同一のウェイタ・リストを有する互いに隣接するピアのノードを捜し出すようにしているのであり、このようにしているのは、別々のノードを再結合することができるのは、それらノードの範囲が互いに隣接しているという条件と、それらノードのウェイタ・リストが互いに同一、或いはいずれも空リストであるという条件との、両方の条件が満足されたときだけだからである。

【００６２】現在の左側ピアにウェイタが存在していなければ、ブロック４１０～４１２と同様にして、ブロック４１７、４１８及び４１９で再結合を実行し、そこから更にこのプロセスは、ブロック４１５へ移行して、その他の左側ピアの有無についてのチェックを実行する。ブロック４１４またはブロック４１５において、その他の左側ピアが発見されなかったことによって、以上の部分に関するプロセスが全て完了したならば、更にこのプロセスは、ブロック４２０において再びセントラル・ノードを取り出し、そしてブロック４０７において右側ピアの有無を調べる。ある右側ピアが発見されたならば、その右側ピアのウェイタと、更にその他の右側ピアの有無と、その「更にその他の右側ピア」のウェイタとを調べた上、左側ピアに関して実行したのと同じシーケンスに従って再結合を実行する。シーケンスが同一であるため、このフローチャートの残りの部分については説明するまでもなく理解されよう。

【００６３】再び説明を図７に戻す。同図において、ノードの分割処理が完了して、プロセスがブロック２１１のサブルーチンへ到達したならば、デッドロック回避アルゴリズムを開始する。このアルゴリズムは、図１４及び図１５に示した形態のものとすることが好ましい。このアルゴリズムは、起動されたならば、みずからの初期化を実行する。この初期化は先ず、ブロック６０１においてデッドロック・スタックをヌル状態にし、次にブロック６０２において、待たせようとしているユーザ（即ち、それを待ち状態とすることによってデッドロックが発生するか否かを調べようとしているユーザ）を、「ユーザA」というポインタに設定し、更にブロック６０３において、この「ユーザA」が現在所有しているロックのうちの先頭のロック（即ち、複数のロックを所有している場合には、それらのロックが１本のキューを形成しているため、そのキューの先頭のロック）を、「ＬＯＣＫ－ＯＦ－Ａ」というポインタに設定する。続いてブロック６０４において、このポインタ「ＬＯＣＫ－ＯＦ－Ａ」を調べ、もし、それがヌル状態である（ＬＯＣＫ－ＯＦ－Ａ＝０）と判断された上に、ブロック６０９においてデッドロック・スタックもヌル・スタックであると判断されたならば、検出の処理は不要であり、それゆえブロック６１０において「デッドロック不在」信号を返送して、このルーチンを完了する。一方、ポインタ「ＬＯＣＫ－ＯＦ－Ａ」によって形成されているキューが、ヌル状態でなかったならば（ＬＯＣＫ－ＯＦ－Ａ≠０）、ブロック６０５において、この「ＬＯＣＫ－ＯＦ－Ａ」のロック範囲を待っているウェイタのうちの先頭のウェイタを、「ユーザB」に設定する。そして、ブロック６０６においてこの「ユーザB」を調べ、それが存在していなかったならば（ユーザB＝０）、ブロック６０７に移行してポインタ「ＬＯＣＫ－ＯＦ－Ａ」を、「ユーザA」のロックのうち、それまでこのポインタが指し示していたロックの次のロックに設定し直した上で、プロセスをブロック６０４へ戻して反復させる。一方、この「ユーザB」が存在していたならば（ユーザB≠０）、ブロック６０８においてポインタ「ＬＯＣＫ－ＯＦ－Ｂ」に、この「ユーザB」が所有しているロックのうちの先頭のロックを設定し、ブロック６１４においてこのポインタ「ＬＯＣＫ－ＯＦ－Ｂ」がヌル・ポインタ（ＬＯＣＫ－ＯＦ－Ｂ＝０）でないかどうかをチェックする。それがヌル・ポインタでなかったならば（ＬＯＣＫ－ＯＦ－Ｂ≠０）、ブロック６１５に移行して「ユーザB」のこの先頭のロックのノードに関するウェイタのうちの先頭のウェイタである「ウェイタX」を、デッドロックのサーチに用いるために特定する。一方、ブロック６１７において、「ウェイタX」が存在していない（ウェイタX＝０）と判断されたならば、ブロック６１６において、このノードに関する次のウェイタを、「ユーザB」に設定した上で、プロセスをブロック６０６へ戻す。一方、ブロック６１７において、あるウェイタが（即ちウェイタXが）検出され、その検出されたウェイタが、ブロック６１８において、「ユーザA」と同一のものである（ウェイタX＝ユーザA）ことが（即ち、「ウェイタXが」オーナーであることが）検出されたならば、ブロック６１９において、「デッドロック検出」信号を返送して、待ち状態を許可しないようにする。一方、それが検出されなかったならば、ブロック６２０において、デッドロック・スタックをチェックし

て、「ウェイタX」が既にデッドロック・スタックに入れられているか否かを判断する。そのスタックに入っていなかったならば、ブロック６２１において「ウェイタX」をデッドロック・スタックへプッシュし、更にブロック６２２において、ポインタ「LOCK－OF－B」が指し示している「ユーザB」のロックに関して待ち状態にある次のウェイタを、新たに「ウェイタX」に設定し、そしてプロセスをブロック６１７に戻して反復させる。このよう、各ノードのウェイタ・リスト中の各々のウェイタを、待ち状態を許可しようとしているウェイタ候補と比較対照して、そのウェイタ候補に実際に待ち状態を許可した場合に、それによってデッドロックが発生するか否かを判断するようにしているのである。もしその待ち状態の許可によって、デッドロックが発生する可能性があるならば、その待ちの許可を阻止し、「デッドロック検出」信号であるリターン信号によって、リクエスタに対して、待ちは許可されないことを通知する。

【００６４】基本的には、デッドロックが発生する可能性があるのは、ある要求されたロック範囲にウェイタが存在しており、そのウェイタが、待ち状態の許可を求められているロック範囲のロックを所有しているとき（例えば、ウェイタX＝ユーザAである場合）である。以上のプロセスは、２レベルのテストを実行するものであり、即ち、リクエスタが所有している複数のロックを個々にチェックし、しかも個々のロックについては、その複数のウェイタに関して次々と実行し、更にその後に、それら複数のウェイタを、それらウェイタが所有している個々のロックについてチェックすることによって、各ウェイタが、待ち状態の許可を求められているロック範囲を、所有していないことを確かめるというものである。

【００６５】以上の、図６～図１５についての説明から分かるように、各々のノードを定義して、サーチの容易なツリー構造を確立するためには、多くのデータが必要である。それら必要なデータの具体的なデータ・セットについては、後に、図２１を参照して説明することにする。ただし、本発明を理解するためには、また、従来技術に対する本発明の顕著な利点を理解するためには、次のことを知っておけば充分である。それは、各ノードは２つのリストすなわちプールを包含しており、その一方は、そのノードのオーナーを表示したオーナー・リストであり、他方は、そのノードに対応した範囲に関する全てのウェイタの表示を包含しているウェイタ・リストだということである。

【００６６】次に図１６について説明する。同図は、本発明の動作を図解して示したものである。時刻Ｔ１において、２つのロック（即ち、Ｌ＃７とＬ＃１０）が存在しているものとする。これらのロックが存在しているため、１～２５の範囲を持ったノードＮ１と、２６～５０の範囲を持ったノードＮ２との、２つのノードの各々に

ついて少なくとも１つずつの、一元化された記述が存在しており、これを、この時刻におけるロック・ツリーの状態を表わした分割状態Ｄ１によって図示してある。各々のノードは、２つずつのリスト、即ちＬリスト（ロック・リスト）とＲリスト（リクエスタ・リスト）とを備えている。ノードＮ１では、Ｌリストは、ロックのオーナーの識別情報（７）を包含しており、また、同じくノードＮ２のＬリストも、そのロックのオーナーの識別情報（１０）を包含している。これらの各ノードのＲリストは空リストであり、そのわけは、この時刻では、待ち要求は存在していないものとしているからである。

【００６７】時刻Ｔ２において、要求Ｒ＃１と要求Ｒ＃３とを受け取ったものとする。これらの要求Ｒ＃１とＲ＃３とのいずれが先に処理されるにせよ、その、先に処理される要求によって、ノードＮ１３が生成されることになる。その理由は、もし要求Ｒ＃１が先に処理されるのであれば、この要求Ｒ＃１の範囲がロックＬ＃７の範囲に対して、その中に含有される「含有」関係にあるために、ノードＮ１３が生成されるし、一方、要求Ｒ＃３が先に処理されるのであれば、この要求Ｒ＃３の範囲がロックＬ＃７の範囲に対して単純オーバラップを成す関係にあるため、やはりノードＮ１３が生成されるからである。要求Ｒ＃１を処理する場合には更に、分割Ｄ２に示すように、ノードＮ１１とノードＮ１２も生成される。ここで特に注意すべきことは、この要求Ｒ＃１がノードＮ１２のリクエスタ・リストの中にウェイタとして載せられているのは、ロックＬ＃７がノードＮ１１、Ｎ１２、及びＮ１３のロック・リストに載せられている間だけであり、それらノードのロック・リストは、この時点では、以前、ノードＮ１において一元化された形で記述されていたロックの、分割された記述を構成しているということである。

【００６８】同様に、ノードＮ１５は、要求Ｒ＃３の範囲が、ロックＬ＃１０の範囲に対して、それを包含する「包含」関係にあったために生成されたノードである。ここでは、この後、時刻Ｔ３になって更なる要求Ｒ＃４２とＲ＃６１とが発生され、更にその後、時刻Ｔ４になって更なる要求Ｒ＃７９が発生されたものとする。これらが発生された後には、リソースの分割状態、並びに各ノードのリストは、図の分割状態Ｄ２に示したようになっている。

【００６９】更にこの後、時刻Ｔ５になって、それまでロックＬ＃１０によってロックされていた範囲２６～５０についてのロック解除を求める、ロック解除要求UNLOCK＃１０が発せられたとする。ノードＮ１４とノードＮ１５とは、それらのウェイタ・リスト（即ちリクエスタ・リスト）が互いに同一であるため、これら２つのノードは再結合されて１つのノードＮ２５になり、分割状態Ｄ３に示すとおりとなる。このとき、ノードＮ１１～Ｎ１３はそれまでの状態がそのまま維持され

るが、それは、ロックＬ＃７が継続しているからである。また、要求Ｒ＃３およびＲ＃７９に対しては、いかなるノードも許可することができず、それは、ノードＮ１３がロックＬ＃７によってロックされたままだからである。一方、要求Ｒ＃４２は、それに関する全ての要求範囲（即ちノードＮ１４及びＮ１５）が、夫々、個別に（原子的に）許可し得る状態になるため、これらのノードＮ１４及びＮ１５に対してロックが許可される（Ｌ＃４２が形成される）。これに関して、これらのノードＮ１４及びＮ１５が再結合されるのが、要求Ｒ＃４２に対してロックが許可されるより前であるのか、それとも後であるかは重要ではなく、更には、仮に、その分割状態を維持するその他の要求が存在していたために、その再結合が不可能であったとしても、それによって何ら不都合は生じない。この具体例においては、要求Ｒ＃４２へのロックが許可される前も後も、ノードＮ１４とノードＮ１５の夫々のウェイタ・リストは互いに同一であり、そのためこれらのノードＮ１４とＮ１５とは、いずれにせよ、再結合されてノードＮ２５になるのである。

【００７０】ここでは更に、時刻Ｔ６になったときに、＃７の範囲（１～２５）のロック解除を求めるロック解除要求ＵＮＬＯＣＫ＃７と、＃４２の範囲のロック解除を求めるロック解除要求ＵＮＬＯＣＫ＃４２とを受け取ったものとする。これらのロック解除要求を受け取ったならば、分割状態Ｄ４に示すように、ノードＮ１１は、フリー・ノード・プールへ戻され、ロック・ツリーから除去され、更に、その他のデータ構造からも除去される。一方、かつてのノード１２に対応しているノードＮ３１は、最先のウェイタからの要求Ｒ＃１に対応してその範囲のロックが許可され（Ｌ＃１が形成され）、また、このとき、かつてのノードＮ１３に対応しているノードＮ３５と、かつてのノードＮ２５に対応しているノードＮ３６との、夫々の範囲のロックを、個別に（原子的に）、しかも同時に、最先の要求Ｒ＃３に対して許可することができる（Ｌ＃３が形成される）。これらのノードＮ３５とＮ３６とは再結合はされない。その理由は、残りのウェイタを表示しているウェイタ・リスト（リクエスタ・リスト）が互いに同一ではないからである。ただし、もし、ロックＬ＃３が解除されるより先に、要求Ｒ＃６１が時間切れになったならば、それらノードは再結合されることになる。

【００７１】本発明のこの実施例では、ロックＬ＃３が解除されたならば要求Ｒ＃６１に対してロックが許可されるようにしてある。しかしながら、このようなアルゴリズムにするか否かは、任意に選択すれば良いことである。即ち、以上に説明した構成に対しては、これとは別のアルゴリズムを組み合わせて実装することも容易であり、それによって例えば、広い範囲を要求するリクエスタにとって時間切れが重大な問題とならない限り、番号の最も大きな一群のリクエスタを優先するようにしたり、或いは、その他の優先権を組み込んだ規則を用いることによって、システムの処理量を可及的に増大させることも可能となる。このような優先権を組み込んだ規則を構築することのできるアルゴリズムは、共用リソースの利用効率を可及的に増大させることができる点において有利なものであるが、しかしながら、現在のところでは、それが必要不可欠と考えられているわけではない。

【００７２】これより図１７～図２０を参照しつつ本発明の好適実施例について説明する。既述の如く、複数種類の状態、ないしは複数種類のルール・セットを、夫々特定のタイプのアクセス要求を受け取ったときに、ロックすべきアドレス可能要素の範囲に対して適用するようにすることも可能である。そして、これが可能であるのは、例えば、共用リソースがメモリであり、アドレス可能要素がバイトであり、またアクセスが、少なくとも、読み書きタイプと、読取りのみのタイプとの２種類であるようにした場合である。更に、これも既述の如く、複数のユーザが、同時に１つのファイルに対して変更を加えることは防止しなければならず、従って、読み書きアクセスは、排他的アクセスとしなければならない。一方、これとは逆に、ファイルを変更することなく読み取ることは、多くのユーザが行なえるようにしておくべきであり、それゆえ、読取りのみのアクセスは、共用アクセスとすべきであろう。従って、その場合には、複数のリクエスタに対して、共用ロック許可するのが良い。この点について付言しておくと、図１６に関連して既に戯明したように、ロックの各々には、オーナー・リスト（ロック・リスト）とウェイタ・リスト（リクエスタ・リスト）とを持たせてある。そして、排他的ロック状態とするためには、既に示したように、オーナー・リストに載せるオーナーをただ１つに限定していたのであるが、共用ロック状態とするためには、このオーナー・リストに、複数のオーナーを載せられるようにすれば良いのである。

【００７３】図１６に示した動作は、次の２点以外は、図１７～図２０に示す本発明の実施例の動作と変わらない。それら２点とは、第１に、ロック状態に関する更なる情報が存在し、この情報も再結合の前に比較参照する必要があること、第２には、ロック状態に応じて、複数のオーナーがオーナー・リストの中に存在することがあるということである。

【００７４】図１７及び図１８を、既に説明した図７及び図８と比較すれば、ロック状態が１種類しかない構成を２つ状態の構成へ拡張するためには、４つのステップを追加するだけで良いことが分かる。それらのステップ以外の部分については、図１７及び図１８は、図７及び図８と同一である。更に詳細に説明すると、図１７においては、「待ち状態が許容可能か否か」を判断する判断ブロック２０６の前に２つの判断ブロック２０６１と２０６２とが挿入され、また、フラグ設定ステップのブロ

ック２０６３が加えられており、更に、判断ブロック２１３１が設けられている。この最後の判断ブロック２１３１は、ロックを要求された範囲が共用ロック範囲であった場合に、デッドロックのチェックステップを実行せずにとばすためのブロックである。以上において、衝突が検出されたならば、ブロック２０６１では、衝突した要求とロックとのうち、ロックの方をチェックして、そのロックが共用ロックであるか否かを判断する。それが共用ロックであったならば、続いて判断ブロック２０６２において、その衝突した要求の方をチェックして、その要求が非排他的要求である（即ち、その要求が排他的ロックを求める要求ではない）か否かを判断する。これら２つの条件が満足されたときに、しかも、これら２つの条件が満足されたときのみに、共用ロックを許可することができ、そしてブロック２０６３において、共用ロックの許可を表わすフラグのセットをすることができる。１つの範囲に関しては、１種類の状態のロックだけ、即ち１つのタイプのロックだけしか許されないため、全てのロックが同一のタイプでなければならない。

【００７５】分割が行なわれた後には、必要とあらば、分割によって生成された範囲をチェックして、もし新たなロックが許可されるとしたら、それが共用ロックであるか否かを判断する。そして、それが実際に共用ロックであり、また、共用ロックであれば許可できるものであれば（例えば、要求された共用ロック範囲と、既に許可されている共用ロック範囲とが同一である場合）、デッドロックが発生するおそれはないのであるから、プロセスを分岐させて、即座に、そのロックを許可した上、そのロックに関するデータを、図７に示したようにしてツリー構造の中に入れることができる。

【００７６】多状態構成の状態数を増やしたい場合には、状態を１種類増やすごとに、一対の判断ブロックを追加するだけで良いことが理解されよう。即ち、それら判断ブロックによって、夫々ロックと要求とを調べて、それらが互いに同一種類の状態であるか否かを調べた上（実際に同一種類であったならば、フラグをセットするようにする）、その追加した状態の種類のロックによって、デッドロックが発生するおそれがあるか否かを判断すれば良い。状態の１種類について一対の判断ブロックを追加すれば良いということは、そのシステム全体の中でどれ程多くの種類のロック状態が利用されているかにかかわりなく、言えることである。

【００７７】次に図１９及び図２０について説明する。これらの図１９及び図２０を、図９及び図１０と比較すれば、多状態に対応可能にするためには、ロック解除のプロセスに、共有オーナーの有無をチェックするステップ（ブロック３３０１）と、当該ロックのオーナーシップを、その共有オーナーに移転するためのステップ（ブロック３３０２）とを付加するだけで良いことが分かる。もし、全ての状態が排他的状態であるならば、これら２つの追加ステップは不要である。

【００７８】既述の如く、本発明を動作させるには、オーナー・リスト（ロック・リスト）とウェイタ・リスト（リクエスタ・リスト）以外にも必要な情報がある。具体例としてのデータ構造のダイアグラムを図２１に示した。図中に記入してある各項の定義は以下のとおりである。

ＬＯＣＫ＿ＥＬＥＭＥＮＴ＿ＰＴＲ；ノードの範囲が関連している対象（即ちロック・オーナー）を示すポインタ。

ＮＥＸＴ＿ＬＯＣＫＥＤ＿ＲＡＮＧＥ；１人のユーザが要求している複数の範囲（割ロック範囲を含む）を表示する要求範囲リストを形成するための、連鎖形ポインタ。

ＮＥＸＴ＿ＴＩＭＥＤ＿ＯＵＴ＿ＮＯＤＥ；ノードが時間切れキューの中に入れられているときに、それを示すスタック・ポインタ。

ＮＥＸＴ＿ＮＯＤＥ＿ＴＯ＿ＲＥＣＯＭＰＯＳ；ノードが再結合キューの中に入れられているときに、それを示すスタック・ポインタ。

ＷＡＩＴ＿ＱＵＥＵＥ＿ＡＮＣＨＯＲ；所与の範囲に関するウェイタ・キューの中の先頭のウェイタを示すポインタ。

ＮＥＸＴ＿ＷＡＩＴＥＲ＿ＯＮ＿ＱＵＥＵＥ；ウェイタ・キューの連鎖を示す連鎖情報。

ＷＡＩＴ＿ＰＥＥＲＳ＿ＣＯＵＮＴ；ノードを分割するときに、全てのサブ範囲に対して個別に（原子的に）許可が与えられようになるために、待ち状態にならねばならないサブ範囲の数。

ＭＡＳＴＥＲ＿ＮＯＤＥ＿ＰＯＩＮＴＥＲ；あるロック範囲を現在所有しているノードを示すポインタ。

ＣＥＮＴＲＡＬ＿ＮＯＤＥ＿ＰＯＩＮＴＥＲ；１つのノードが複数のサブ範囲へ分割されているときに、それら全てのサブ範囲についての大域的情報を包含している１つのノードを示すポインタ。

ＯＦＦＳＥＴ；範囲の始点。

ＲＡＮＧＥ＿ＬＥＮＧＴＨ；始点から測った、範囲の大きさ。

ＰＡＲＥＮＴ　ＮＯＤＥ；ツリー中の、所与のノードの上位のノード。

ＬＥＦＴ＿ＰＥＥＲ；ある１つのノードが分割されて作られた複数のサブ範囲のうち、所与の範囲より若い番号の側に位置しているサブ範囲。

ＲＩＧＨＴ＿ＰＥＥＲ；ある１つのノードが分割されて作られた複数のサブ範囲のうち、所与の範囲より大きい番号の側に位置しているサブ範囲。

ＬＥＦＴ＿ＣＨＩＬＤ；当該ノードの始点よりも、その全域が小さい番号の側に位置している範囲を表示しているノード。

ＲＩＧＨＴ＿ＣＨＩＬＤ；当該ノードの終点よりも、そ

の全域が大きい番号の側に位置している範囲を表示しているノード。

ＳＵＢＲＡＮＧＥ＿ＳＴＡＲＴ；ノードそれ自体の範囲の始点（そのノードがオリジナル・ノードであれば、オフセットに等しい）。

ＳＵＢＲＡＮＧＥ＿ＥＮＤ；ノードそれ自体の範囲の終点（即ちオフセット＋長さ－１）。

尚、図中のフラグは、ロック状態のアイデンティティ（ＩＤ）を表示するためと、上で説明した、その他の種々の目的のためとに使用するデータである。システムにおいて使用可能な状態が２種類だけである場合には、フラグ・ビットとしては１つのビットがあれ良く、それによって、２種類の状態を区別することができる。更に、多状態の実施例では、オーナー・キュー情報と、キューの上の次オーナーを示すポインタとが、図２１に示したデータの中に含まれており、それによって、図１９に示したオーナーシップの判断（ブロック３３０１）と、オーナーシップの変更（ブロック３３０２）とを容易にしている。

【００７９】以上に示したデータ構造、並びに具体的なデータ項目は、単に具体例を示したものに過ぎず、これとは異なったデータの組み合わせを用いることも可能であることを理解されたい。更には、ノードに対応した記述子の各々に関する具体的なデータも、リストを指定するポインタの形とすることが好ましい。これらのリストは、先入れ先出し（ＦＩＦＯ）レジスタと、ターム・スタックとして構成することが好ましいことも判明している。これに関して、順序は重要でなく、更には、ここで使用しているキューも、そのように形成することが好ましいが、ただし、これらの細部構成は、本発明の動作にとっては絶対的なものではなく、単なる好都合な手段であるに過ぎない。

【００８０】以上の説明から分かるように、ここで提供している、共用リソースのためのロック管理システムは、ロック・データの複製という作業を不要にしており、また、ロック・データ情報を保持するためのメモリの専用部分を大きく必要としないために、ハードウェア効率の高いものとなっている。また、ロック・データ情報を、サーチの容易なツリーデータ構造として実装することができるため、高速で機能することができ、更には、実際に要求されたアドレス可能要素の範囲以上の共用リソースをロックする必要がないため、共用リソースの利用効率を高めることができるものとなっている。この最後の点に関しては、要求された範囲よりも多い個数のアドレス可能要素をロックするように本発明を構成した場合には、それによって特に利点が得られるものではなく、共用リソースの利用度は最適から一歩後退することになるが、それもまた、本発明の概念並びに範囲から逸脱するものではないことを理解されたい。また、そのようにすることが、有用な場合もある。いずれにせよ、ロックの単独記述を分割して、オーバラップしない異なったロックの単独記述にし、またそれら複数の単独記述を再結合して１つの単独記述にするという特徴によって、共用リソースを最適に利用することが可能となっているのである。

【００８１】以上に本発明の１つの好適実施例について説明したが、当業者であれば容易に理解できるように、本発明の概念並びに範囲から逸脱することなく、種々の変更態様として本発明を実施することが可能である。

【図面の簡単な説明】

【図１】本発明を実装することのできる、ＩＢＭ社のＳ／３７０ＬＡＮファイル・サーバ・モデルのブロック図である。

【図２】本発明の一実施例に係るロック機構に使用している、ロック・ノードのツリー構造の模式図である。

【図３】オーバラップが発生したときにロック要求を分割するノードの分割操作を示した、図２のツリー構造の模式図である。

【図４】包含形のオーバラップが発生したときのノードの分割操作を示した、図２のツリー構造の模式図である。

【図５】含有形のオーバラップが発生したときのノードの分割操作を示した、図２のツリーの模式図である。

【図６】本発明の一実施例に係るロック機構の、基本的な論理動作を示したフローチャートである。

【図７】本発明の一実施例に係るロック機構の、範囲ロック処理サブルーチンの論理動作を示したフローチャートである。

【図８】本発明の一実施例に係るロック機構の、範囲ロック処理サブルーチンの論理動作を示した、図７のフローチャートに続くフローチャートである。

【図９】本発明の一実施例に係るロック機構の、範囲ロック解除処理サブルーチンの論理動作を示したフローチャートである。

【図１０】本発明の一実施例に係るロック機構の、範囲ロック解除処理サブルーチンの論理動作を示した、図９のフローチャートに続くフローチャートである。

【図１１】本発明の一実施例に係るロック機構の、ノード再結合の処理の論理動作を示したフローチャートである。

【図１２】本発明の一実施例に係るロック機構の、ノード再結合の処理の論理動作を示した、図１１のフローチャートに続くフロー・チャートである。

【図１３】本発明の一実施例に係るロック機構の、ノード／範囲の分割の処理の論理動作を示したフローチャートである。

【図１４】本発明の一実施例に係るロック機構の、デッドロック回避の処理の論理動作を示したフローチャートである。

【図１５】本発明の一実施例に係るロック機構の、デッ

ドロック回避の処理の論理動作を示した、図１４のフローチャートに続くフローチャートである。

【図１６】本発明の一実施例に係るデータ構造の、そのノードに対応した情報の構成を示した模式図である。

【図１７】ロック状態が多状態である場合に対応し得るように変更した、図７及び図８の範囲ロック処理サブルーチンの変更例のフローチャートであり、図７の部分に対応したフローチャートである。

【図１８】ロック状態が多状態である場合に対応し得るように変更した、図７及び図８の範囲ロック処理サブルーチンの変更例のフローチャートであり、図８の部分に対応したフローチャートである。

【図１９】ロック状態が多状態である場合に対応し得るように変更した、図９及び図１０の範囲ロック解除処理

サブルーチンの変更例のフローチャートであり、図１０の部分に対応したフローチャートである。

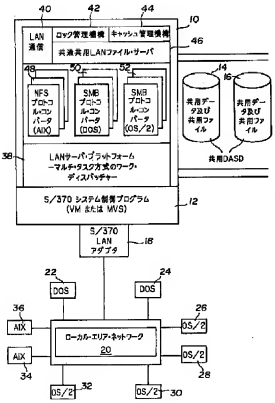
【図２０】ロック状態が多状態である場合に対応し得るように変更した、図９及び図１０の範囲ロック解除処理サブルーチンの変更例のフローチャートであり、図９の部分に対応したフローチャートである。

【図２１】本発明の動作と各ノードを定義するために使用するデータとを説明するための一覧表である。

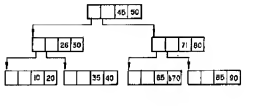
【符号の説明】

１０　　メインフレーム・コンピュータ
１２　　システム制御プログラム
１４、１６　共用データ及び共用ファイル（ＤＡＳＤ）
４６　　共用ＬＡＮファイル・サーバ・アプリケーション

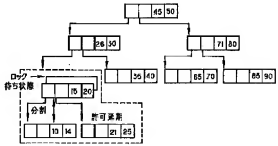
【図１】



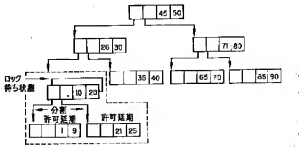
【図２】



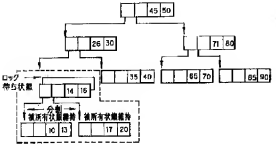
【図３】



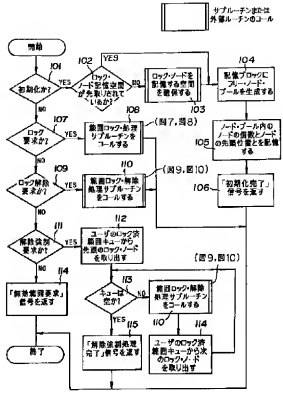
【図４】



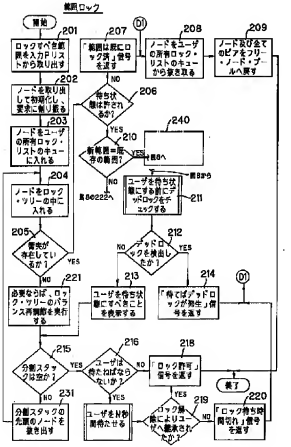
【図５】



【図６】



【図７】

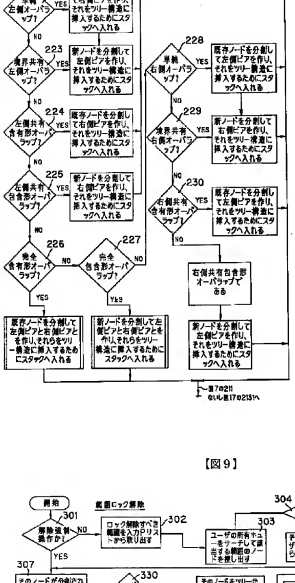


【図２１】

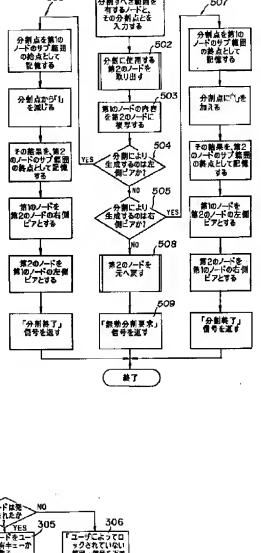
ノードの全定義

0	OWNER_TASK_ID		
8	LOCK_ELEMENT_PTR	NEXT_LOCKED_RANGE	
16	NEXT_TIMED_OUT_NODE	NEXT_NODE_TO_RECOMPOS	
24	WAIT_QUEUE_ANCHOR	NEXT_WAITER_ON_QUEUE	
32	WAIT_PEERS_COUNT	MASTER_NODE_POINTER	
40	CENTRAL_NODE_POINTER	OFFSET	
56	RANGE_LENGTH	PARENT_NODE	
64	LEFT_PEER	RIGHT_PEER	
72	LEFT_CHILD	RIGHT_CHILD	
80	SUBRANGE_START	SUBRANGE_END	
88	reserved	BF	flags
96			

【図８】

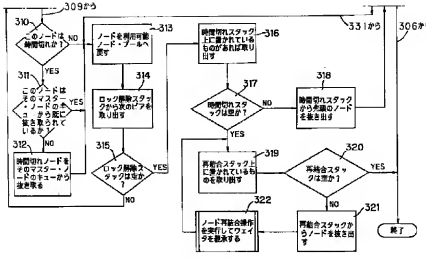


【図１３】

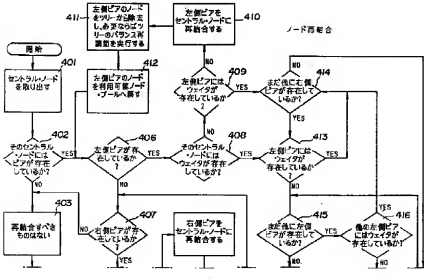


【図９】

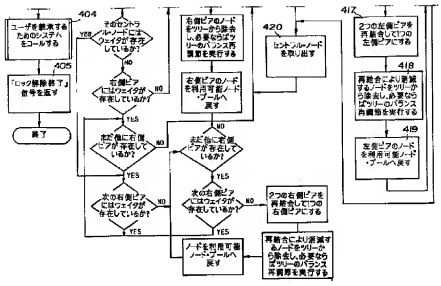
【図１０】


309から
310
ノードは時間切れか
NO
YES
311
このノードはそのマスター・ノードの子ノードから既に取り去られているか
YES
NO
312
時間切れノードをその下マスター・ノードのキューから取り去る
313
ノードを利用可能ノード・プールへ戻す
314
ロック解除スタックから次のビアを取り出す
315
ロック解除スタックは空か
YES
NO
316
時間切れスタックに置かれているものがあれば取り出す
317
時間切れスタックは空か
NO
YES
318
時間切れスタックから兄弟のノードを取り出す
319
再結合のスタックに置かれている印を取り出す
320
再結合スタックは空か
YES
NO
321
再結合スタックからノードを取り出す
322
ノード再結合操作を実行してウェイクを除去する
331から
306から
終了

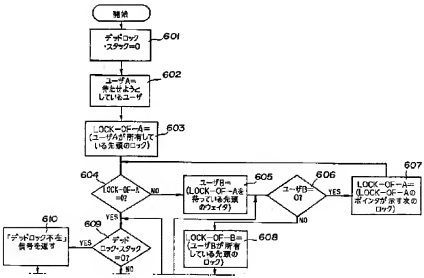
【図１１】


開始
401
セントラル・ノードを取り出す
402
そのセントラル・ノードにはビアが存在しているか
YES
NO
403
再結合すべきものは無い
411
左側ビアのノードをリーダが調査しあるならばリーダのバランス再調整を実行する
412
左側ビアのノードを利用可能ノード・プールへ戻す
406
左側ビアが存在しているか
YES
NO
407
右側ビアが存在しているか
NO
YES
左側ビアをセントラル・ノードに再結合する
410
左側ビアをセントラルノードに再結合する
ノード再結合
409
左側ビアには・ウェイクが存在しているか
NO
YES
408
そのセントラル・ノードには・ウェイクが存在しているか
YES
NO
414
さらに他に側ビアが存在しているか
NO
YES
413
左側ビアには・ウェイクが存在しているか
YES
NO
415
さらに他に側ビアが存在しているか
YES
NO
416
他の左側ビアには・ウェイクが存在しているか
YES
NO

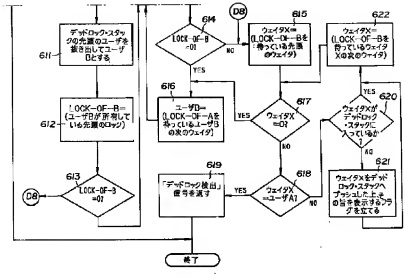
【図１２】


404
ユーザを継承するためのシステムコールする
405
「ロック解除終了」信号を返す
終了
その左ノードがツリーのルートノードにか、ウェイダが存在していないか?
YES
NO
右側ピアには ウェイダが存在していないか?
NO
右側ピアのノードをツリーから除き、必要ならばツリーのバランス再調整を実行する
右側ピアのノードを利用可能ノード・プールへ戻す
NO
YES
まだ他にお隣ピアが存在していないか?
NO
まだ他にお隣ピアが存在していないか?
NO
YES
次のお隣ピアにはウェイダが存在していないか?
NO
次のお隣ピアにはウェイダが存在していないか?
NO
YES
YES
ノードを利用可能ノード・プールへ戻す
2つの左側ピアを再結合してひとつの左端ピアにする
再結合により派生するノードをツリーから除き、必要ならばツリーのバランス再調整を実行する
420
コントロールノードを取り出す
417
418
419
友告ピアのノードを利用可能ノード・プールへ戻す

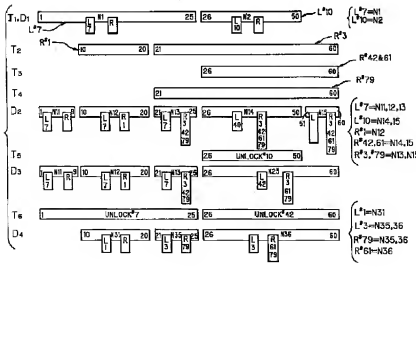
【図１４】


開始
デッドロック・スタック=0
601
ユーザA=自分をいじっているユーザ
602
LOCK-OF-A=(ユーザAが所有している先頭のロック)
603
604
LOCK-OF-A=nil
NO
YES
ユーザB=(LOCK-OF-Aを待っている先頭のウェイダ)
605
ユーザB=01
606
YES
NO
607
LOCK-OF-A=(LOCK-OF-Aのポインタが示す次のロック)
610
「デッドロック不生」信号を返す
609
デッドロック・スタック=0?
YES
NO
LOCK-OF-B=(ユーザBが所有している先頭のロック)
608

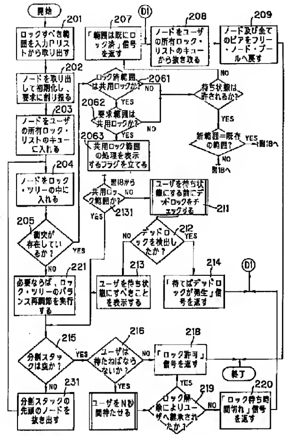
【図１５】


D8
614
LOCK-OF-B =01
NO
YES
615
ウェイタX= (LOCK-OF-Bを待っている先頭のウェイタ)
622
ウェイタX= (LOCK-OF-Bを待っているウェイタX0及びウェイタ)
611
デッドロック・スタンダの先頭のユーザを抜き出してユーザDとする
616
ユーザD= (LOCK-OF-Aを持っているユーザBの次のウェイタ)
617
ウェイタX =0?
YES
NO
620
ウェイタXがデッドロック・スタンダに入っているか
YES
NO
612
LOCK-OF-B= (ユーザDが所有していた先頭のLOCK)
613
LOCK-OF-B =0?
D8
619
"デッドロック検出"信号を送す
618
ウェイタX =ユーザA?
YES
NO
621
ウェイタXをデッドロック・スタンダへプッシュしたことの旨を表示するフラグを立てる
終了

【図１６】


T1,D1
T2
T3
T4
D2
T5
D3
T6
D4
L*7=N1
L*10=N2
R*1
R*3
R*42&61
R*79
L*7=N11,12,13
L*10=N14,15
R*1=N12
R*42,61=N14,15
R*3,*79=N13,N15
UNLOCK*10
UNLOCK*7
UNLOCK*42
L*1=N31
L*3=N35,36
R*79=N35,36
R*61=N36

【図17】


開始
201
ロックすべき範囲を入力Pリストから取り出す
202
ノードを取り出して初期化し、要求に割り振る
203
ノードをユーザの所有ロック・リストのキューに入れる
204
ノードをロック・ツリーの中に入れる
205
衝突が存在しているか？
YES
NO
221
必要ならば、ロック・ツリーのバランス再調節を実行する
215
分割スタックは空か？
YES
NO
231
分割スタックの先頭のノードを抜き出す
207
「範囲は既にロック済」信号を返す
D1
208
ノードをユーザの所有ロック・リストのキューから抜き取る
209
ノード及び全てのピアをフリー・ノード・プールへ戻す
ロック済範囲は共用ロックか？
2061
2062
要求範囲は共用ロックか？
2063
共用ロック範囲の処理を表示するフラグを立てる
持ち状態は許されるか？
新範囲＝既存の範囲？
図18へ
図18から
共用ロック範囲か？
2131
ユーザを持ち状態にする前にデッドロックをチェックする
211
デッドロックを検出したか？
212
213
ユーザを持ち状態にすべきことを表示する
214
「待てばデッドロックが発生」信号を返す
216
ユーザは待たねばならないか？
218
「ロック許可」信号を返す
終了
ユーザをN秒間待たせる
219
ロック解除によりユーザへ継承されたか？
220
「ロック待ち時間切れ」信号を返す
YES
NO

【図18】

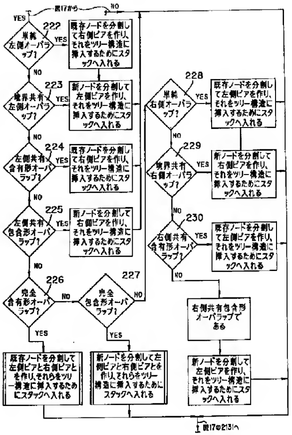
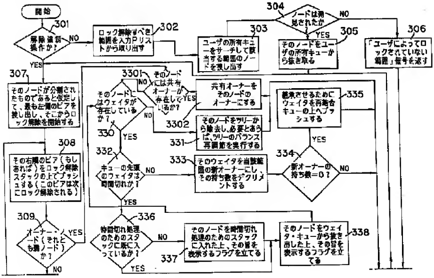
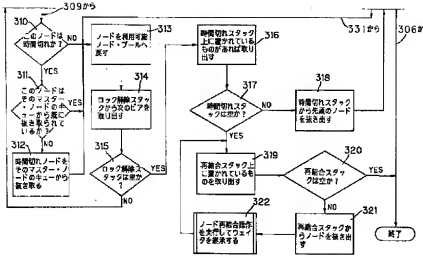


図17から
YES
NO
222
単純左側オーバラップ？
既存ノードを分割して右側ピアを作り、それをツリー構造に挿入するためにスタックへ入れる
223
境界共有左側オーバラップ？
新ノードを分割して左側ピアを作り、それをツリー構造に挿入するためにスタックへ入れる
224
左側共有含有形オーバラップ？
既存ノードを分割して右側ピアを作り、それをツリー構造に挿入するためにスタックへ入れる
225
左側共有包含形オーバラップ？
新ノードを分割して右側ピアを作り、それをツリー構造に挿入するためにスタックへ入れる
226
完全含有形オーバラップ？
227
完全包含形オーバラップ？
既存ノードを分割して左側ピアと右側ピアとを作り、それらをツリー構造に挿入するためにスタックへ入れる
新ノードを分割して左側ピアと右側ピアとを作り、それらをツリー構造に挿入するためにスタックへ入れる
228
単純右側オーバラップ？
既存ノードを分割して左側ピアを作り、それをツリー構造に挿入するためにスタックへ入れる
229
境界共有右側オーバラップ？
新ノードを分割して右側ピアを作り、それをツリー構造に挿入するためにスタックへ入れる
230
右側共有含有形オーバラップ？
既存ノードを分割して左側ピアを作り、それをツリー構造に挿入するためにスタックへ入れる
右側共有包含形オーバラップである
新ノードを分割して左側ピアを作り、それをツリー構造に挿入するためにスタックへ入れる
図17の2131へ
YES
NO

【図19】


開始
301
解除要請操作か？
302
ロック解除すべき範囲を入力Pリストから取り出す
303
ユーザの所有キューをサーチして該当する範囲のノードを捜し出す
304
ノードは発見されたか
305
そのノードをユーザの所有キューから抜き取る
306
「ユーザによってロックされていない範囲」信号を返す
307
そのノードが分割されたものであると仮定して、最も左側のピアを捜し出し、そこからロック解除を開始する
308
その右側のピア（もしあれば）をロック解除スタックの上でプッシュする（このピアは次にロック解除される）
309
オーナー・ノード（それとも別ノード）か？
3301
そのノードには共有オーナーが存在しているか？
共有オーナーをそのノードのオーナーにする
そのノードにはウェイタが存在しているか？
330
3302
そのノードをツリーから除去し、必要とあらば、ツリーのバランス再調節を実行する
331
332
キューの先頭のウェイタは時間切れか？
333
そのウェイタを当該範囲の新オーナーにし、その待ち数をデクリメントする
継承させるためにウェイタを再割合キューの上へプッシュする
335
334
新オーナーの待ち数＝0？
336
時間切れ処理のためのスタックに既に入っているか？
337
そのノードを時間切れ処理のためのスタックに入れた上、その旨を表示するフラグを立てる
そのノードをウェイタ・キューから抜き出した上、その旨を表示するフラグを立てる
338
YES
NO

【図２０】


309から
310
この ノードは 切断 切れか ?
ND
YES
311
この ノードは その マスター ノードの ホーム キューから 既に 取られて いるか ?
YES
NO
312
待機 切れ ノードを その マスター ノードの キューから 取り取る
313
ノードを 利用 可能 ノード・プールへ 戻す
314
ロック 解除 スタックから 次の ビアを 取り出す
315
ロック 解除 スタックは 空か ?
YES
NO
316
待機 切れ スタック 上に 置かれて いる もの が あれば 取り出す
317
待機 切れ スタックは 空か ?
NO
YES
318
待機 切れ スタック から 先頭 の ノードを 抜き 出す
319
再結合 スタック 上に 置かれて いる もの を 取り 出す
320
再結合 スタック は 空か ?
YES
ND
321
再結合 スタック から ノードを 抜き 出す
322
ノード 再結合 指令 を 実行 して ウェイブ を 要求 する
331から
306から
終了

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 04-319734

(43)Date of publication of application : 10.11.1992

(51)Int.Cl. G06F 9/46
G06F 12/00
G06F 13/00
G06F 15/16

(21)Application number : 04-016794

(22)Date of filing : 31.01.1992

(71)Applicant : INTERNATL BUSINESS MACH CORP <IBM>

(72)Inventor : HART CASWALL A

(30)Priority

Priority number : 91 660451 Priority date : 22.02.1991 Priority country : US

(54) LOCK MANAGEMENT DEVICE

(57)Abstract:

PURPOSE: To provide a method and device which can efficiently cope with plural mutually overlapping lock requests that lock ranges of addressable elements in a common resource.

CONSTITUTION: When locks L#7 and L#10 accept new lock requests R#1, 3, 42, 61, and 79 overlapping an existent locked range to nodes N1 and N2, division is performed to generate nodes N11-N14 and further generate a node N15. At this time, a lock list L of the nodes N11-N13 is 7 as it is and a lock list of the node N14 is 10 as it is; and the L#7 and L#10 are held as they are, but 1, 3, 42, 79, 3, 42, 61, 79, 3, 42, 61, and 79 are registered in a requester list R of nodes N12-N15. Once an unlock request UNLOCK#10 is received, the nodes N14 and N15 are recoupled together to generate a node N25, and L#42 (not overlapping with L#7) is generated for the node. Similarly, when UNLOCKs #7 and #42 are requested, nodes are reconnected and nonoverlapping L#1 and L#3 are generated.